Hacia la Inteligencia Artificial con Amstrad

AMSTRAD
ESPAÑA



Jeremy Vine

Hacia la inteligencia artificial con Amstrad

Dedicatoria

A Christine

'From quiet homes and first beginning,
   Out to the undiscovered ends,
There's nothing worth the wear of winning,
   But laughter and the love of friends.'

Hilaire Belloc (1870-1953)

Hacia la inteligencia artificial con Amstrad

Jeremy Vine



HACIA LA INTELIGENCIA ARTIFICIAL CON AMSTRAD

Edición española de la obra

ON THE ROAD TO ARTIFICIAL INTELLIGENCE: AMSTRAD CPC464
Jeremy Vine

publicada en castellano bajo licencia de

Shiva Publishing Limited
64 Welsh Row
Nantwich, Cheshire CW5 5ES
Inglaterra

Traducción:
Miguel A. Revilla
Dpto. de Matemática Aplicada
Facultad de Ciencias
Universidad de Valladolid

INDESCOMP, S. A.
Avda. del Mediterráneo, 9
28007 Madrid

ISBN: 28-86176-28-X
Depósito Legal: M. 10950-1985

Impresión: Gráficos Lormo. Isabel Méndez, 15. 28038-Madrid.

Producción de la edición española:
Vector Ediciones
Gutierre de Cetina, 61
28017 Madrid

Contenido

Prólogo

Los ordenadores están pasando rápidamente a formar parte de la vida cotidiana. El desarrollo de la tecnología del silicio ha representado un avance tal que ahora está a nuestro alcance lo que no hace mucho era pura ciencia ficción. Se están desarrollando máquinas que pueden trabajar con escalofriante precisión, imitando el modelo humano en cuanto a pensamiento y conocimiento. Esta evolución ha provocado la aparición de una de las industrias más prósperas: los micros domésticos. Para algunos son simples juguetes, pero la potencia intrínseca de un ordenador doméstico es grande, y existe la posibilidad de crear programas que simulen procesos del pensamiento humano.

La programación interactiva se concentra sobre cierto aspecto del lenguaje BASIC: las instrucciones responsables de manipular información. El libro enseña cómo usar estas órdenes y las combina en programas que conversan con el usuario. Se presenta también el creciente campo de la inteligencia artificial, una de las áreas más excitantes de la informática. Se supone que el lector tiene escasos conocimientos de programación. El libro está escrito para personas sin apenas conocimientos de BASIC, y sólo supone que el lector ha jugado con unas cuantas órdenes elementales, como PRINT e INPUT. Por otra parte, por si su experiencia en programación fuera nula, hemos incluido un curso de choque de BASIC al final del texto. En sí mismo, el libro es una introducción al BASIC, pero, a diferencia de otros, intenta presentar la noción de inteligencia artificial y enseñar al usuario cómo escribir programas que interaccionen con él. Se describen dos programas de simulación que convierten el Amstrad en un compañero con el cual mantener bis a bis una conversación inteligente y que se comporta de una manera similar a un ser humano, incluso con sus defectos.

Con un pequeño esfuerzo usted escribirá programas que le proporcionarán horas de placer y al mismo tiempo una idea de lo que puede esperar de su Amstrad.

Para terminar con una nota personal, debo agradecer la ayuda, el ánimo

y el apoyo que he recibido de familiares y amigos, y de todo el personal de Shiva Publishing. A ellos, y a todas las personas que han despertado mi interés en este área, se debe en parte este libro.

Londres, 1984
Jeremy Vine

Sobre el autor

Jeremy Vine fue educado en la William Ellis Grammar School, Highgate, Londres, y en el City of London Polytechnic, donde concluyó estudios de psicología. Jeremy tuvo su primer contacto con los ordenadores mientras estudiaba para su graduación y ya nunca ha estado lejos de ellos. Cuando dejó el politécnico, trabajó por libre para la revista Acorn User, antes de incorporarse a Acorn Computers Limited. Durante este período también estudiaba para un Master de Ciencias en Neurofisiología. Ahora, tras abandonar Acorn, Jeremy dedica todo su tiempo a trabajar como escritor independiente. Además de sus libros para Shiva, colabora regularmente con varias revistas de informática, de las que es consultor. Sus aficiones para el tiempo libre son tenis, piano, fotografía y, por supuesto, ordenadores domésticos.

1

¿Pueden pensar las máquinas?

La ciencia ficción suele llegar a hacerse realidad; así como era imposible creer, hace sólo unas décadas, que el hombre pisaría alguna vez la Luna, es aún increíble pensar en máquinas que puedan contestar y actuar de forma inteligente. Sin embargo el día en que la inteligencia artificial llegue a ser un hecho está amaneciendo; no pasará mucho tiempo antes de que un ordenador pueda escribir estas mismas palabras. Sin embargo, antes de que usted comience a pensar que su Amstrad puede actuar como un ser humano, quisiera puntualizar que la inteligencia artificial para ordenadores domésticos no está aún lo bastante avanzada. Pero esto no es razón para no intentar llegar tan lejos como se pueda.

El propósito de este libro es triple. En primer lugar, mostrarle diversas instrucciones en lenguaje BASIC y enseñarle a escribir programas que trabajen únicamente con textos y manipulen los introducidos por el usuario. No se supone más que un conocimiento superficial de BASIC, pero si el lector no tiene suficiente confianza para abordar el Capítulo 3, en el que se comienza a programar, puede consultar el apéndice A, donde hemos escrito una guía relámpago de BASIC y sus principios básicos. En segundo lugar, se muestra cierto número de técnicas que usted puede incorporar a sus propios programas para hacerlos más interactivos; finalmente, se desarrollan programas que capacitan al usuario para comunicarse libremente con el ordenador. ¿Qué quiero decir con comunicarse libremente? Observe la siguiente interacción entre un humano y un ordenador:

ORDENADOR:	ENCANTADO DE CONOCERTE, JEREMY ¿POR QUE QUIERES HABLAR CONMIGO?
HUMANO:	TENGO SUEÑOS TERRIBLES
ORDENADOR:	¿APARECEN OTRAS PERSONAS EN ELLOS?
HUMANO:	SI, MI HERMANA

ORDENADOR:	PARECE UNA RELACION COMPLICADA
HUMANO:	ESTOY DEPRIMIDO POR ESA RAZON
ORDENADOR:	¿CON QUE FRECUENCIA TE SIENTES DEPRIMIDO?
HUMANO:	CONTINUAMENTE
ORDENADOR:	ENTIENDO
HUMANO:	Y SUEÑO A MENUDO
ORDENADOR:	TE ENVIDIO ... YO NO PUEDO SOÑAR
HUMANO:	¿NO TENGO QUE PENSAR EN ELLA?
ORDENADOR:	YO LO HARIA SI ESTUVIERA EN TU LUGAR
HUMANO:	¿QUIERES DECIR QUE ES NORMAL?
ORDENADOR:	NO PUEDO DECIR LO QUE PIENSO
HUMANO:	TU NO PIENSAS. NO ERES MAS QUE UNA MAQUINA
ORDENADOR:	¿TE PREOCUPAN LAS MAQUINAS?
HUMANO:	CLARO QUE NO. LOS ORDENADORES NO TIENEN CEREBRO
ORDENADOR:	LOS ORDENADORES SOMOS PERFECTOS

Esta conversación no tiene lugar en un ordenador enorme, sino en el Amstrad. Es un ejemplo de lo que se puede realizar con los micros domésticos; escribiremos el programa correspondiente en otro lugar del libro. Quizás esté pensando que ya tenemos una máquina inteligente, pues puede mantener una conversación como la anterior. Bueno, sintiéndome como un mago al que se le ha caído una carta de la manga, tengo que admitir que se trata de un truco. Ciertamente, el programa permite al usuario comunicarse libremente con el ordenador, y éste hace réplicas inteligentes, pero ¿significa esto que la máquina esté pensando?

Para cuando lleguemos al final de este libro espero haberle inducido a reflexionar sobre si las máquinas pueden o no pensar, lo cual es una cuestión controvertida entre los investigadores en el área de la inteligencia artificial. Es difícil dar una definición de AI (inteligencia artificial) a causa de la cantidad de argumentos y controversias que rodean el tema. Quizás sirva para describir cualquier acción realizada por un ordenador y que previamente sólo podía ser ejecutada por humanos. O puede describir el procedimiento que la máquina está usando para realizar una tarea, como jugar al ajedrez.

Le dejo con este problema para que medite sobre él. Cuando avancemos a lo largo del libro y aprendamos las diferentes formas de comunicarnos con el ordenador, y él, a su vez, de comunicarse con nosotros, trate de decidir por usted mismo en qué grado su Amstrad es una máquina pensante, si es que lo es en alguno.

2

Creación de un programa interactivo

La elaboración de un programa interactivo es una combinación de muchos factores. Comienza como una idea y evoluciona a lo largo de diversas etapas hasta que está completo y a plena satisfacción del autor. El plan subyacente a un programa es tan importante como la forma en que está escrito; la clasificación de problemas en una etapa precoz facilita el desarrollo. Este capítulo considera los principios generales que es preciso tener en mente antes de escribir un programa, más aún si se trata de un programa interactivo.

La programación interactiva consiste simplemente en la escritura de programas que puedan intercambiar información con el usuario. Lo notable es el uso de BASIC para escribir programas que permiten al usuario comunicarse libremente con el ordenador; en posteriores capítulos veremos cómo usar una amplia variedad de instrucciones en lenguaje BASIC que harán que nuestro Amstrad llegue a tener la apariencia de un ser pensante, e incluso inteligente. Lo que intentaremos conseguir es enseñar a nuestro ordenador a pensar.

En la creación de un programa existirá siempre una idea o un propósito que lo origina. Debería meditarse cuidadosamente cuáles son los objetivos exactos del programa y la forma en que trabajará. ¿Qué quiere usted que el usuario vea en la pantalla? ¿Qué le permitirá teclear? Usted debe plantearse muchas cuestiones sobre el programa si quiere que todo vaya bien.

Por ejemplo, explicaremos algunas de las cosas que un usuario puede hacer para embarullarlo todo. Lo que proponemos es una forma de incorporar al programa, para potenciar su utilidad, rutinas que prevengan incluso lo peor a lo que ha de enfrentarse un ordenador: ¡el usuario! El ordenador no tiene modo de conocer o entender lo que el usuario desea y es en todos los sentidos ciego al mundo exterior. Los ordenadores son tontos (pero no

le digan esto después a Sigmund, o se enfadará) y tienen que ser alimentados con información.

La información es el fluido vital de la máquina; los ordenadores necesitan una dieta saludable de datos crudos. Este libro se centra en un aspecto de la programación, que es la manipulación de texto, sea éste generado por el ordenador o introducido por el usuario. Usted puede que ya conozca algunas instrucciones de BASIC y sea capaz de introducir (INPUT) y escribir (PRINT) información, pero hay muchas más cosas que se pueden realizar con texto. BASIC ofrece un conjunto de instrucciones que permiten al programador manipular texto para crear la impresión de una máquina pensante. Aspectos a tener en cuenta cuando se escribe un programa son:

1. Usar diagramas de flujo para facilitar la planificación del programa. (Hay varios diagramas de flujo en el libro que ayudan a entender la forma en que trabajan los programas.)
2. Planificación de lo que el usuario va a ver en pantalla.
3. Facilidad de uso. Intente hacer el programa lo más fácil de usar posible.
4. Detección de errores. Asegúrese de que los errores involuntarios no afecten a la marcha normal del programa.

Si alguno de estos puntos parece no tener sentido por el momento, no se inquiete. Los entenderemos a medida que vayamos avanzando; al final del libro estos principios habrán quedado grabados indeleblemente en usted.

Recuerde siempre que estamos tratando con una caja vacía. Nuestra tarea es llenar esta caja y transformarla en una cueva de Aladino para cualquiera que se asome a ella. Pero si vamos a ser creativos y escribir programas interactivos, tendremos que programar. Vayamos directamente a ello. ¡Felices interacciones!

3

Cadenas literales

Ya hemos dicho que nos disponemos a enseñar a pensar a nuestro Amstrad. La primera etapa de este proceso va a ser el manejo de entradas. En este capítulo y en el próximo veremos cómo el ordenador acepta información y la manipula según nuestras instrucciones. Pero recordemos qué se entiende por «variable».

Programa 3.1

```
10 INPUT nombre$
20 INPUT edad
30 PRINT nombre$
40 PRINT edad
```

El programa 3.1 contiene dos variables: una variable literal (*string*), señalada por un $ (dólar) al final del nombre, llamada nombre$, y una variable numérica llamada edad. Cuando se ejecuta el programa, en respuesta al primer interrogante ? podemos teclear caracteres alfanuméricos (en otras palabras, cualquier cosa que queremos) y esta respuesta será almacenada en nombre$.

Al pulsar la tecla ENTER aparece otro signo de interrogación. Esta vez el Amstrad está esperando que se teclee un número; su valor se almacenará en edad. Las líneas 30 y 40 escriben el contenido de nombre$ y a continuación el de edad. Usted sin duda ha adivinado que el programa le está preguntando su nombre y edad. Pero esto no está tan claro para el usuario, pues todo lo que ve al ejecutarlo son dos signos de interrogación !. Vale, mejoremos el programa:

Programa 3.2

```
10 INPUT"nombre: ",nombre$
20 INPUT"edad: ",edad
30 PRINT nombre$;" TIENE";edad;" A\OS"
```

Al ejecutar este programa usted puede ver que los interrogantes son más explícitos, pues ahora indicamos al usuario que introduzca su nombre y edad. Vuelva a ejecutar el programa, pero esta vez conteste con su nombre a ambas cuestiones. Comprobará que la máquina almacena la primera respuesta correctamente, puesto que se trata de una cadena, es decir, de caracteres cualesquiera. Sin embargo, la segunda variable, edad, requiere una respuesta numérica, y por tanto conservará el valor 0 ya que no hemos introducido datos numéricos. El Amstrad rechaza en esta situación los caracteres alfabéticos y solicita que se vuelva a introducir la información.

Tras haber recordado brevemente las variables, es hora de jugar un poco con cadenas (de caracteres).

MANIPULACIÓN DE CADENAS 1: LEFT$

Puesto que uno de los principales objetivos de este libro es centrarnos en la introducción de texto y su posterior manejo por el ordenador, repasemos algunas instrucciones del BASIC del Amstrad que nos facilitan la manipulación de cadenas. Pero antes de nada, consideremos el siguiente problema: necesita un programa que escriba las letras de mi nombre, añadiendo una letra en cada línea de la siguiente forma:

J
JE
JER
JERE
JEREM
JEREMY

Podríamos escribir un programa como el que sigue:

Programa 3.3

```
10 PRINT"J"
20 PRINT"JE"
30 PRINT"JER"
40 PRINT"JERE"
50 PRINT"JEREM"
60 PRINT"JEREMY"
```

Pero no llegaremos muy lejos si tenemos que ser tan explícitos con el ordenador. Lo que ocurre con el programa 3.3 es que estamos escribiendo cada cosa que ha de aparecer en la pantalla, lo que no es precisamente una forma de ahorrar tiempo al escribir el programa. Por supuesto, hay un modo más rápido, y el BASIC del Amstrad posee las instrucciones necesarias para ayudarnos en esta situación.

Lo que necesitamos es una forma de leer cualquier cadena dada y escribirla letra a letra, aumentando un carácter cada línea como en el ejemplo anterior, sin necesidad de todas esas instrucciones PRINT. «¿Cómo puedo hacer esto?», le oigo preguntar. La respuesta es sencilla. Usaremos la instrucción LEFT$.

El siguiente programa usa LEFT$ para escribir el primer carácter de la cadena introducida por el usuario. Pruébelo.

Programa 3.4

```
10 INPUT"Escribe una palabra: ",palabra$
20 PRINT LEFT$(palabra$,1)
```

Ahora cambie el número 1 en la línea 20 por otros valores para ver qué ocurre. Variando este valor, el número de caracteres tomados, de izquierda a derecha, de la palabra será mayor o menor. Eche un vistazo a la figura 3.1.

Nuestra siguiente tarea es producir el mismo efecto sin tener que utilizar una instrucción PRINT para cada línea. Introduzca en su ordenador el programa 3.5:

Programa 3.5

```
10 frase$="EL VELOZ ZORRO MARRON"
20 contador=1
30 PRINT LEFT$(frase$,contador)
40 contador=contador+1:GOTO 30
```

Ejecute el programa y ...?! Ahora el problema radica en que hemos aumentado el valor de contador cada vez, y cuando excedemos del número de caracteres de la cadena continuamos escribiendo únicamente frase$. Cambie la línea 40 por:

```
40 contador=contador+1:IF contador>19 THEN END E
LSE 30
```

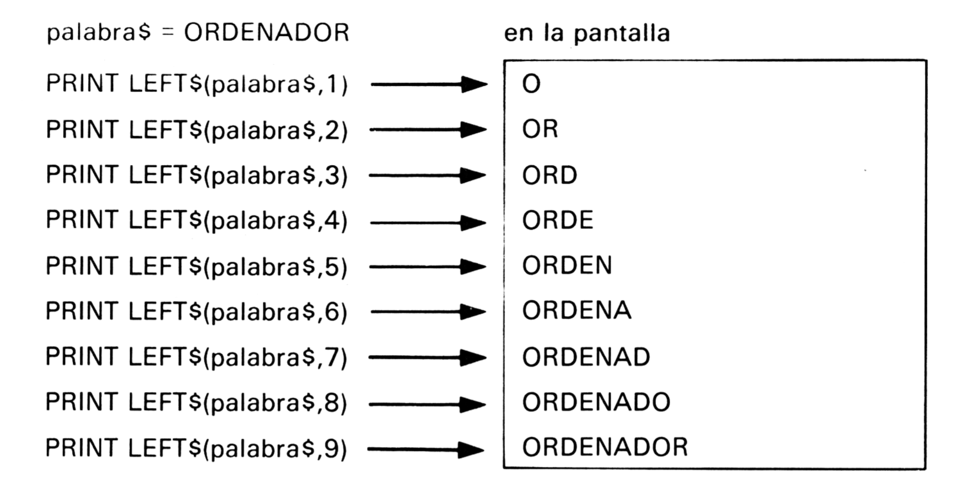


Figura 3.1 Efecto de LEFT$.

¡Por fin el efecto deseado! ¿O aún no? Hay todavía un problema. Para parar el programa cuando todos los caracteres han sido escritos, es preciso contar el número de caracteres que tiene frase$ (incluyendo los espacios); ésta es la razón de la instrucción IF contador>19 de la línea 40.

Sin embargo, no siempre conoceremos de antemano el número de caracteres de la cadena. Una vez más, el BASIC del Amstrad viene al rescate con una instrucción que permite al ordenador determinar la longitud de una cadena dada.

MANIPULACIÓN DE CADENAS 2: LEN

La instrucción LEN proporciona esa información. En el programa 3.6 usamos LEN para obtener el número de caracteres contenidos en entrada$. Introduzca primero una palabra; después vuelva a ejecutar el programa e introduzca una pequeña frase.

Programa 3.6

```
10 MODE 2
20 INPUT"escriba una palabra o frase: ",entrada$
30 PRINT"El numero de caracteres de esa cadena e
s";LEN(entrada$)
```

Si teclea unas cuantas palabras observará que LEN cuenta los espacios exactamente igual que los demás caracteres. Los espacios son, pues, tenidos en

cuenta en este ejemplo, y es importante tener esto presente, ya que más tarde echaremos un vistazo más detenido a los espacios entre palabras. Pero volvamos a LEN.

En el programa 3.5 nuestro problema era que no conocíamos el número de caracteres de la cadena. Reescribamos el programa usando LEN:

Programa 3.7

```
10 INPUT frase$
20 FOR contador=1 TO LEN(frase$)
30 PRINT LEFT$(frase$,contador)
40 NEXT
```

Observe que he introducido un bucle FOR-NEXT. Si aún no ha utilizado la instrucción FOR-NEXT, un vistazo al manual del usuario le explicará sus posibles aplicaciones. Dicho brevemente, el bucle varía desde el primer carácter de frase$ hasta la longitud total de la cadena, que ha sido determinada por LEN en la línea 20.

El programa 3.7 no sólo es más eficiente que el programa 3.5, sino también más rápido. Esto puede ser de suma importancia en programas más grandes en los que se lleven a cabo muchos procesos. Pero dejemos esto para más adelante. Es el momento de introducir otro par de instrucciones BASIC.

MANIPULACIÓN DE CADENAS 3: RIGHT$ Y MID$

Si ha entendido cómo funciona LEFT$ (izquierda), la siguiente instrucción le resultará muy fácil. RIGHT$ (derecha) es, como su nombre indica, una instrucción de manejo de cadenas; por si no lo ha adivinado todavía, realiza la misma función que LEFT$, sólo que a la inversa. Bueno, ¡no exactamente al revés! Volvamos al programa 3.4, pero cambiando LEFT$ por RIGHT$ en la línea 20:

Programa 3.8

```
10 INPUT"Escriba una palabra: ",palabra$
20 PRINT RIGHT$(palabra$,1)
```

RIGHT$ no trabaja del todo como usted pensaba. Con el valor 1 en la línea 20, RIGHT$ da el último carácter de la cadena. Sin embargo, pruebe con otros valores. El resultado no es de atrás hacia delante. ¿Está claro? Observe la figura 3.2.

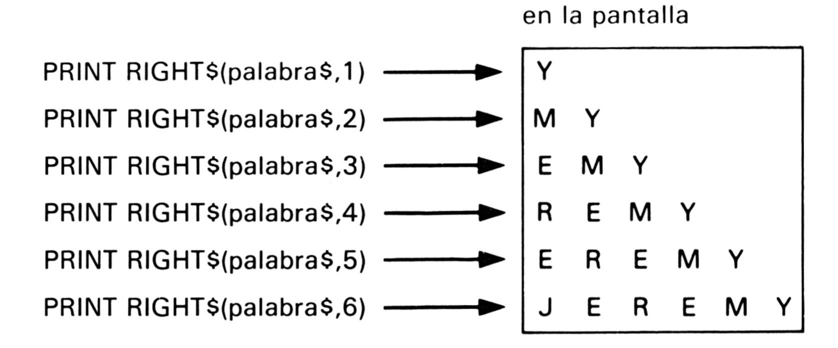


Figura 3.2 Efecto de RIGHT$.

La razón por la que digo que puede no estar claro es que se podría esperar que el resultado de la instrucción RIGHT$ (palabra$,3) fuese YME; en otras palabras, que escribiese la cadena comenzando por la derecha y hacia la izquierda, pero, como puede ver en la figura 3.2, esto no es así. Si aún no está seguro acerca del uso de RIGHT$, pruebe los programas 3.4, 3.5 y 3.7 sustituyendo LEFT$ por RIGHT$ cada vez que aparezca.

Examinemos ahora MID$. Al igual que LEFT$ y RIGHT$, es una función literal; pero, a diferencia de estas instrucciones, lleva asociados tres argumentos; por ejemplo: MID$ (ejemplo$,X,Y). X e Y son dos de los argumentos y los explicaré por orden. Supongamos que ejemplo$ contiene la cadena INTERACCIÓN. Así es como trabaja MID$:

El primer argumento es igual que el LEFT$ y RIGHT$, o sea, el nombre de la variable literal que va a ser tratada; en nuestro caso, ejemplo$. El segundo valor actúa como un indicador de la posición de partida. Por tanto, el valor 6 significa que la cadena que vamos a extraer comienza en esta posición, ocupada en este caso por la letra A.

Finalmente, el tercer valor (representado por Y) indica el número de caracteres que se deben tomar a partir de la posición de X (incluido el carácter que está en esa posición). Como muestra la figura 3.3, se toman las primeras seis letras desde la posición X (ocurre que sólo quedan seis caracteres en esta cadena). Pongamos esto en un programa:

Programa 3.9

```
10 CLS
20 ejemplo$="INTERACCION"
```

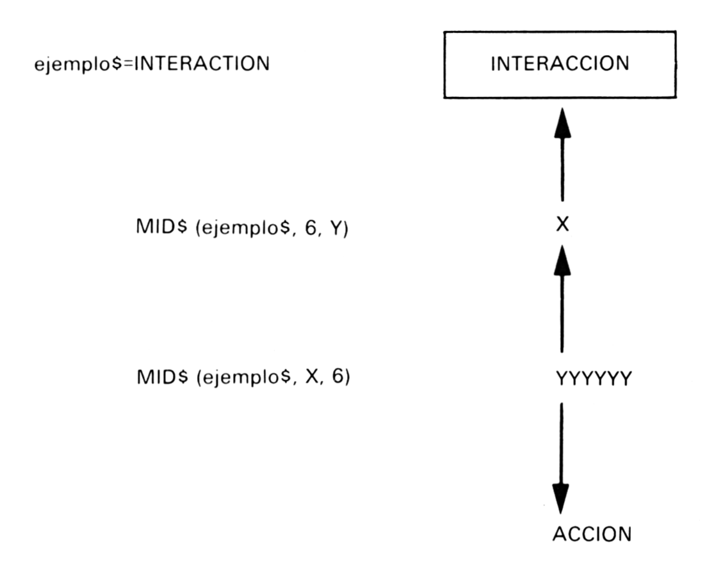


Figura 3.3 Función de MID$.

```
30 INPUT"Donde quiere poner el indicador (es dec
ir, la X)?: ",x
40 INPUT"Numero de caracteres a partir de esa po
sicion: ",y
50 PRINT MID$(ejemplo$,x,y)
```

Ejecute el programa introduciendo diferentes valores y vea cómo los valores de x e y alteran la palabra impresa por la línea 50. (Nótese que he utilizado letras mayúsculas y minúsculas para x e y, al objeto de demostrar que esto es indiferente para el ordenador; de hecho, podría combinar ambas formas y aún serían aceptadas.) La función MID$, junto con las otras funciones literales antes mencionadas, es extremadamente útil; volveré sobre ellas más adelante. Dejemos por un momento la manipulación de cadenas para fijar nuestra atención en otro aspecto del manejo de las entradas.

MENÚES

No, no estoy hablando de comida. He mencionado en el Capítulo 2 que una parte importante de un programa interactivo es la visualización en pantalla. Parte fundamental de esta exposición es con frecuencia un menú de

opciones entre las que el usuario debe elegir, de modo que vamos a comenzar escribiendo el menú de opciones para nuestro próximo programa. Teclee el programa, respetando los números de línea que aquí aparecen, pues vamos a desarrollar un programa en varias etapas.

Programa 3.10a

```
50 CLS
60 PRINT"C O D I G O S   A S C I I"
70 PRINT,,"1 Codigo ASCII a caracter"
80 PRINT,,"2 Caracter a codigo ASCII"
90 PRINT,,,,"Elija (1-2)"
100 a$=INKEY$
110 a=VAL(a$):IF a<1 OR a>2 THEN 100
120 PRINT a
130 ON a GOTO 30,40
140 RETURN
```

Añada también las siguientes líneas:

```
10 MODE 1: ZONE 40
20 GOSUB 50
```

Ejecute el programa. Como no está completo, provocará el mensaje de error NO SUCH LINE (no hay tal línea) si pulsa el número 1 o el 2. Ejecútelo unas cuantas veces y observe que sólo responde a esos dos números. Si teclea cualquier otro número, el ordenador lo ignora. De momento no se preocupe de cómo trabaja el programa; yo le iré explicando cómo funciona esta parte, y también el resto del programa, en lo que queda de este capítulo y en el siguiente.

Una cosa que no debería haberle pasado inadvertida es el extraño título que puede ver en la pantalla: CODIGOS ASCII. ¿Quién o qué es ASCII? Descubrámoslo.

ASCII

ASCII (pronúnciese AS-KI) son las siglas de *American Standard Code for Information Interchange* (Código americano estándar para el intercambio de la información). Intente decir esto varias veces. ¡No es de extrañar que se abrevie! Pero, ¿qué significa todo esto?

Como seguramente habrá descubierto ya, todos los ordenadores son diferentes; de BASIC, por ejemplo, hay muchas versiones implementadas y

todas ellas tienen peculiaridades. Una de las pocas normas adoptadas por la mayor parte de los fabaricantes de ordenadores es el uso del conjunto de caracteres ASCII. ASCII es un código usado por el ordenador para representar caracteres y códigos de control. Para nuestros propósitos, nos centraremos en el juego de caracteres del teclado. Si no lo ha hecho ya, grabe con SAVE el programa menú y a continuación teclee el programa 3.11.

Programa 3.11

```
1 REM ia-17
10 FOR ascii=32 TO 126
20 PRINT CHR$(ascii);
30 NEXT
```

Este programa escribe los caracteres representados por el valor ASCII correspondiente. Seguramente habrá observado que he utilizado una instrucción de BASIC, CHR$, en el programa 3.11. Para explicar el efecto de CHR$, cargue con LOAD el programa 3.10*a* otra vez y añada lo siguiente:

Programa 3.10b

```
240 CLS:INPUT"Escriba un codigo ASCII. Pulse ENT
ER: ",codigo
250 PRINT
260 PRINT codigo;"es el codigo ASCII de ´";CHR$(
codigo);"´"
270 PRINT,,,,"Pulse ´S´ para continuar"
280 a$=INKEY$:IF a$="s" OR a$="S" THEN 290 ELSE
280
290 RETURN
```

Añada también la línea

```
30 GOSUB 240
```

Ahora grabe el programa y ejecútelo. Se le pedirá que introduzca un código ASCII numérico. Teclee cualquier número entre 32 y 126 (éstos son los códigos que representan el conjunto de caracteres). La conversión del número ASCII en un carácter es realizada por la función CHR$ de la línea 260.

La acción de CHR$ es producir un carácter a partir de un número dado. Este número es el valor ASCII del carácter; es decir, si usted introduce 65, la respuesta será A, porque el valor ASCII 65 es el código del carácter A.

De acuerdo. ¿Y cómo se convierte un carácter en un número ASCII? Esto

se puede hacer con la función ASC. La utilizaremos al final de nuestro programa. Asegúrese de que tiene el programa cargado (LOAD) en su Amstrad antes de continuar:

Programa 3.10c

```
170 CLS:PRINT"Introduzca un caracter: "
180 a$=INKEY$:IF a$<>"" GOTO 190 ELSE 180
190 PRINT"El codigo ASCII de ´";a$;"´ es"ASC(a$)
200 PRINT,,"Pulse ´S´ para continuar"
210 a$=INKEY$:IF a$="S" OR a$="s" GOTO 220 ELSE
210
220 RETURN
```

Escriba también la línea

40 GOSUB 170: GOSUB 50

y cambie la línea 30 a

30 GOSUB 240: GOSUB 50

Ahora el programa está completo. Grábelo (SAVE) y después ejecútelo (RUN). Elija la opción 2 e introduzca un carácter cualquiera. La línea 190 efectúa la conversión del carácter que está contenido en la cadena a$. ASC(a$) da el código ASCII del carácter contenido en a$. Usted puede ver cómo funciona tecleando la siguiente línea:

PRINT ASC("J")

Cambiando el carácter se producirá un código diferente.

¿Qué ocurre cuando a$ contiene más de un carácter? Bien, la función ASC sólo da el valor del primer carácter. Se estará seguramente preguntando si he olvidado explicar los demás elementos del programa. ¡Tranquilo! Las restantes instrucciones las explicaré en el próximo capítulo. De momento observe en la figura 3.4 un diagrama de flujo de los mecanismos del programa.

Vea si puede descubrir qué defecto tiene el programa. Recuerde lo que dije en el capítulo 2 acerca de prever todas las posibles entradas que pueda hacer el usuario. Volveré sobre este programa en el capítulo 6, dedicado a la detección de errores. ¡Creo que tiene suficientes pistas por ahora!

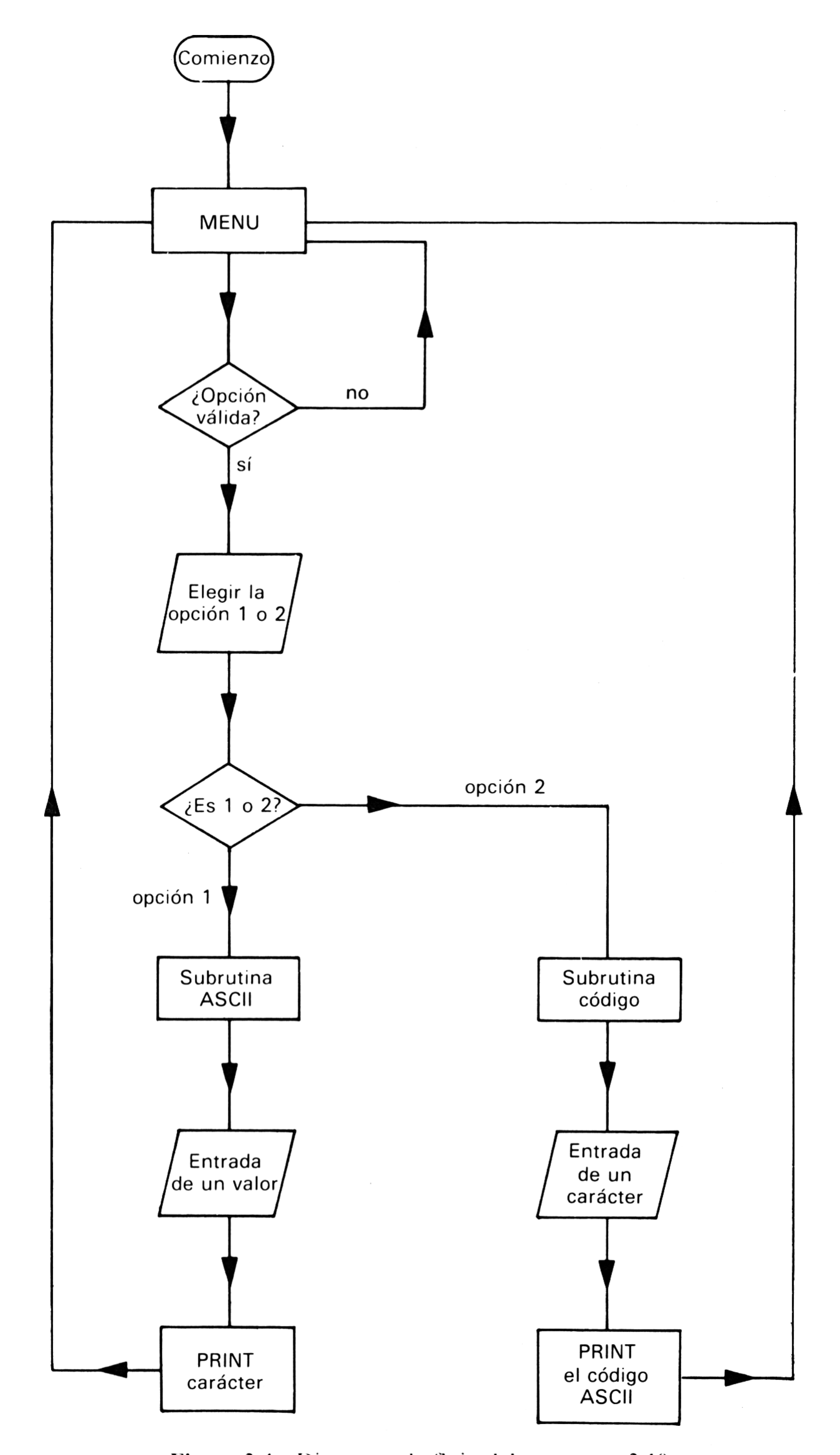


Figura 3.4 Diagrama de flujo del programa 3.10.

4

Más cadenas

Un aspecto importante del desarrollo de un programa interactivo es asegurarse de que el ordenador lea todo lo que el usuario teclee. Usted puede pensar que ya hemos dejado este punto resuelto con la instrucción INPUT, pero no es así. Para ver lo que quiero decir, escriba el programa 4.1:

Programa 4.1

```
10 CLS
20 GOSUB 100:INPUT primerafrase$
30 GOSUB 100:LINE INPUT segundafrase$
40 PRINT,,"primerafrase$ = ´";primerafrase$;"´"
50 PRINT,,"segundafrase$ = ´";segundafrase$;"´"
90 END
100 PRINT,,"Teclee lo siguiente:","MI AMSTRAD ES
 UN GRAN ORDENADOR"
120 RETURN
```

Ejecute este programa y podrá ver que ambas variables contienen la misma frase que hemos introducido. No obstante, ejecútelo otra vez, pero insertando en esta ocasión una coma en la frase tal como se indica a continuación:

MI AMSTRAD, UN GRAN ORDENADOR

¿Ve la diferencia? Cuando usamos INPUT en la línea 20, la variable retendrá sin problemas el contenido íntegro de la cadena, siempre que no haya ninguna coma en la frase. Pero cuando se inserta una coma, INPUT rechaza toda la cadena introducida. Para resolver este problema el BASIC del Amstrad posee la instrucción LINE INPUT, que acepta cualquier cosa que

se teclee. He puesto LINE INPUT en la línea 30 del programa; cuando ejecute el programa comprobará que todo lo que usted teclee, a pesar de las comas, es aceptado.
LINE INPUT no sólo acepta comas, sino que también reconoce espacios a la izquierda. Con esto quiero indicar que si usted teclea unos cuantos espacios antes de su frase, LINE INPUT incluirá esos espacios iniciales en segundafrase$, mientras que INPUT no lo hará. Ensaye escribiendo una frase con espacios por delante para comprobar la diferencia.

Pero, de todos modos, ¿por qué todo este enredo? Bueno, el desarrollo de nuestros programas interactivos tiene por objeto permitir al usuario comunicarse libremente con el ordenador. Para ello el usuario debe gozar de libertad para teclear cualquier cosa que desee, y es muy probable que teclee alguna frase que contenga una coma. Si el ordenador tiene que analizar la entrada del teclado debe ser capaz de entender todo lo que se haya tecleado, y esto no es siempre posible con la instrucción INPUT, como acabamos de ver. LINE INPUT asegura que todo lo tecleado se almacena en una determinada cadena.

RECONOCIMIENTO DE UNA PALABRA FAMILIAR: INSTR

Ahora vamos a afrontar otro problema que nos surge en la creación de un programa interactivo: cómo reconocer una determinada palabra dentro de una secuencia de caracteres. LINE INPUT puede asegurar que toda palabra tecleada sea captada, pero ¿cómo localizar una palabra concreta en una frase? No es tan difícil como pudiera parecer de entrada. Usted ya estará imaginando que alguna instrucción del Amstrad servirá para resolver este problema. La instrucción es INSTR.

Lo que INSTR hace es buscar en una cadena (en este caso, en una frase larga que hemos introducido) una palabra determinada. Cuando ha encontrado esta palabra (o grupo de caracteres) INSTR determina la posición del primer carácter de la palabra dentro de la cadena.

Aclarémoslo con un ejemplo. Teclee el programa 4.2.

Programa 4.2

```
10 CLS:ZONE 40
20 PRINT"Escriba una frase"
30 LINE INPUT a$
40 b$="BASIC"
50 encuentro=INSTR(a$,b$)
60 PRINT,,"Valor de la variable ´encuentro´ = ";
encuentro
```

```
70 PRINT,,"Usted escribio: ",;a$
80 IF encuentro=0 THEN 120 ELSE 90
90 PRINT SPC(encuentro-1);"^"
100 PRINT"La coincidencia se encuentra aqui"
110 END
120 PRINT,,"La cadena ´";b$;"´ no ha sido encont
rada":END
```

Si ahora teclea usted algún viejo refrán, lo probable es que reciba el mensaje «The string BASIC was not found» («La palabra BASIC no ha sido encontrada»). Esto se debe a que el programa está comprobando la existencia de una cadena, en este caso la cadena alfanumérica BASIC que se ha asignado a b$ en la línea 30. Ejecute de nuevo el programa y teclee lo siguiente:

ESTE ES UN BASIC DIFERENTE

Verá cómo esta vez la cadena coincide con una palabra contenida en la frase. El valor de la variable «encuentro» es la posición en que el principio de la palabra BASIC se encuentra dentro de la frase tecleada. La línea del programa que hace todo el trabajo es la 50. INSTR(a$,b$) genera un valor. Si localiza una cocincidencia; es decir, si la variable b$ está en la cadena introducida como a$, entonces INSTR determina la posición y asigna ese valor a la variable numérica «encuentro». En cambio, si no hay coincidencias, «encuentro» sigue siendo 0. La línea 80 utiliza este resultado y de acuerdo con él imprime el mensaje apropiado. Vea en la figura 4.1 un diagrama de flujo del programa.

INSTR es una instrucción muy útil y, como veremos más adelante, desempeña un papel crucial en la búsqueda de palabras clave o frases que el ordenador pueda entender. Más adelante volveremos a este tema. Ahora retrocedamos un poco para ver algunas instrucciones que he utilizado en programas de este capítulo y del capítulo 3 sin la debida explicación.

PRESENTACIÓN EN PANTALLA 1: INKEY$

Si usted ejecutó el programa menú del capítulo 3, recordará que todo estaba preparado para el usuario y, en la medida de lo posible, bastaba con pulsar una tecla para continuar con el programa. Una parte importante de todo programa es la presentación de información en la pantalla y las facilidades que se dan al usuario para que introduzca los datos. Cuanto más favorables sean estos aspectos, más interactivo resultará el programa. Ya he utilizado algunas instrucciones orientadas a este fin; ahora voy a explicarlas.

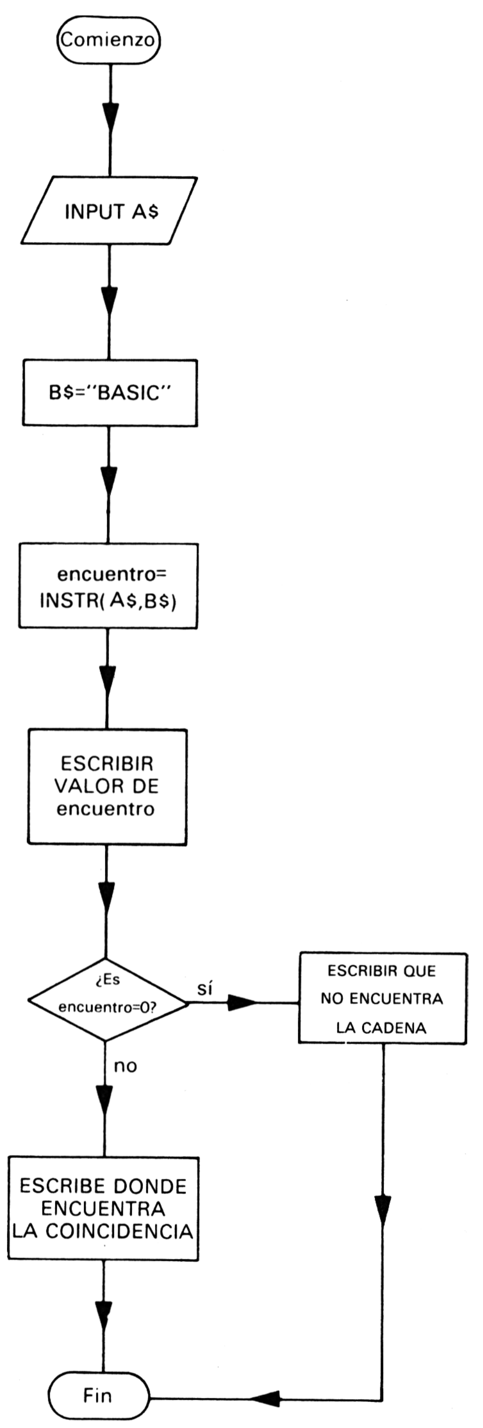


Figura 4.1 Diagrama de flujo.

Vuelva al programa 3.10 y observará que he usado en las líneas 120 y 190 la instrucción INKEY$. Cuando se ejecuta esta instrucción, el ordenador examina el teclado para ver si alguna tecla está siendo pulsada. Si es así, se asigna el carácter correspondiente en la variable que figura a la izquierda de signo igual:

Programa 4.3

```
10 caracter$=INKEY$:IF caracter$="" THEN 10
20 PRINT"Ha tecleado ";caracter$
```

Si no hay ninguna tecla pulsada, INKEY$ da la cadena vacía. Para asegurar de que el programa espere hasta que se pulse una tecla, el programa vuelve a línea 10. Sin embargo, no es necesario utilizar posteriormente la variable almacenada, como yo he hecho en la línea 20 del programa 4.3. La instrucción se puede usar también con objeto de introducir una pausa en el programa hasta que se pulse una tecla. Pruebe el programa 4.4:

Programa 4.4

```
10 CLS
20 PRINT"Estoy esperando que pulse una tecla"
30 a$=INKEY$:IF a$="" THEN 30
40 PRINT"Ya era hora!"
```

La línea 10 borra la pantalla y el programa se queda en la línea 30 hasta que se pulse una tecla. Así pues, INKEY$ proporciona métodos de examinar directamente el teclado para ver si se está pulsando alguna tecla y, en tal caso, determinar cuál es esta tecla.

PRESENTACIÓN EN PANTALLA 2: ZONE, TAB Y SPC

Hay dos instrucciones que se pueden utilizar para escribir espacios dentro de una instrucción PRINT. Se trata de TAB y SPC. TAB se usa combinada con la instrucción PRINT. Significa «tabulación» y permite al usuario especificar la posición a la que quiere llevar el cursor en una sentencia PRINT.

TAB lleva un argumento asociado. Así, PRINT TAB(5); moverá el cursor a la columna número 5 de la línea actual. SPC actúa de una forma parecida, pero no necesita el signo de punto y coma al final de la instrucción. La figura 4.2 muestra ejemplos de las instrucciones TAB y SPC y sus efectos en la pantalla del monitor.

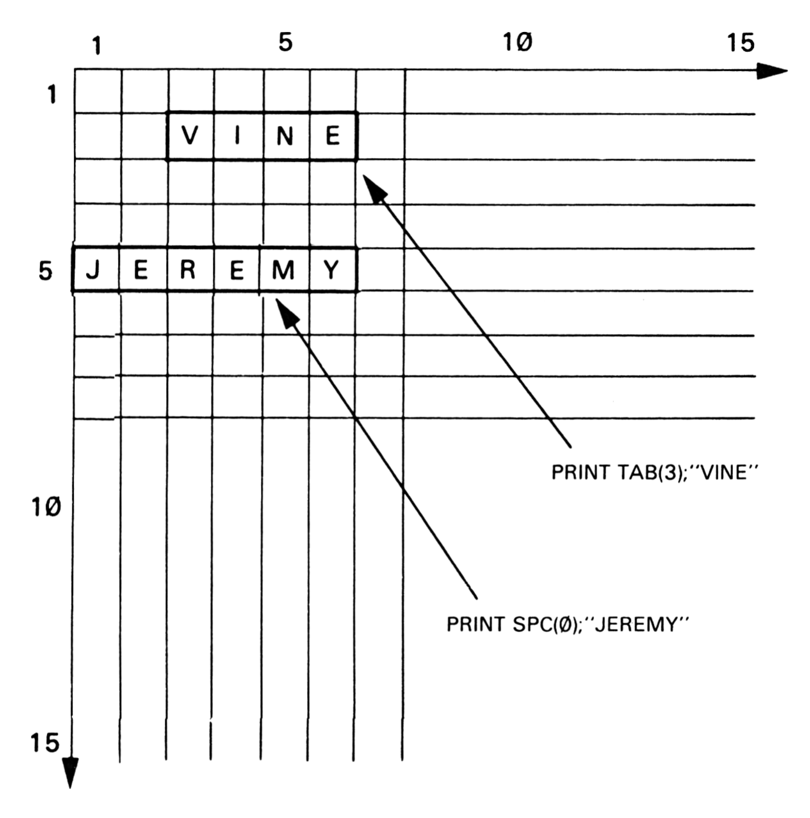


Figura 4.2 Efecto de las instrucciones TAB y SPC.

He utilizado SPC en el programa 4.2 para intercalar el número correcto de espacios en una línea antes de escribir una palabra. Esto asegura que las palabras queden alineadas una debajo de otra. (Vuelva atrás y ejecute el programa 4.2 si no recuerda este detalle.) SPC y TAB son dos funciones muy útiles que facilitan la construcción rápida de pantallas bien presentadas.

El lector habrá notado que he usado el comando ZONE, y volverá a verlo muchas veces en este libro. ZONE se encarga de cambiar la anchura de la «zona» en la que se escribe cuando se usa la instrucción PRINT. Así, cuando usamos una coma para saltar a la siguiente zona de escritura (es decir, PRINT), podemos decir al ordenador dónde empieza esta zona dando su anchura como argumento de ZONE. En el caso de nuestros programas, normalmente hacemos que el programa interprete las comas que siguen a una instrucción PRINT como «escribe una línea en blanco», pues hemos puesto la instrucción ZONE 40. Recuerde, sin embargo, que el número que pongamos después de ZONE para conseguir este efecto depende del modo de pantalla que estemos usando. Como el modo 1 escribe en 40 columnas, necesitamos dar saltos de 40 espacios. Obviamente, cambiaremos este número

de acuerdo con el modo en que estemos trabajando y los efectos que deseemos conseguir.

Ya hemos visto cómo trabaja ASCII y la importancia que puede tener la conversión entre variables numéricas y alfanuméricas. Como usted sabe, una variable numérica no puede leer una entrada alfanumérica. No obstante, hay una forma de convertir una cadena para que pueda ser leída.

CONVERSIÓN DE CADENAS: VAL y STR$

Consideremos un problema. Yo deseo introducir por el teclado un número utilizando una variable alfanumérica y convertirla después de tal manera que una variable numérica pueda manejar el dato introducido. ¿Podemos hacerlo? No tiene mérito adivinar que esto es posible; el programa 4.5 muestra cómo:

Programa 4.5

```
10 a$=INKEY$:IF a$="" THEN 10
20 a=VAL(a$)
30 IF a<1 OR a>5 THEN 10
40 PRINT a
50 ON a GOTO 100,110,120,130,140
```

El programa no está completo y por consiguiente se estrellará en la línea 50. La línea 10 espera por el dato; en la línea 20 es donde tiene lugar la conversión. La función VAL (significa «valor») opera sobre a$, que debería contener un número, y lo t ransforma en un número real que asigna a la variable numérica «a». Para cerciorarse de que se ha introducido un número perteneciente a un determinado intervalo, la línea 30 comprueba que el valor de «a» no está fuera del margen especificado. Si el número es incorrecto, el programa vuelve a la línea 10 en espera de que se introduzca un número correcto. El programa 4.5 es la base de la subrutina GOSUB 50 del programa 3.10*a* del capítulo 3. VAL convierte variables literales en variables numéricas; la acción inversa la realiza la función STR$. Teclee el programa 4.6:

Programa 4.6

```
10 numero=1.2
20 PRINT numero*2
30 a$=STR$(numero)
40 PRINT a$*2
```

Para demostrar que el número 1.4 ha sido transformado a una variable literal, la línea 20 realiza una función matemática sobre una variable numérica y, por supuesto, funciona. Sin embargo, al llegar a la línea 40, el ordenador genera un mensaje de error porque la máquina está intentando efectuar una multiplicación con una variable literal. En la línea 30, STR$ ha convertido la variable numérica en una cadena, asignando la cadena resultante a a$. Para comprobar el contenido de a$ teclee PRINT a$.

Estas funciones del BASIC del Amstrad dan mayor flexibilidad a su programación y, como le he demostrado con el programa menú, pueden ser muy útiles.

OPCIONES EN LOS PROGRAMAS: ON

El programa 4.5 incluye una instrucción en la que quiza no se haya fijado: ON. Esta instrucción permite alterar el orden de ejecución de un programa, saltando a una línea específica dependiendo del valor de cierta variable. Por ejemplo, en el programa menú, en función del número introducido el programa salta a la rutina adecuada. Por tanto, una línea tal como

ON tecla GOTO 220, 250, 800

significa que si el valor numérico de «tecla» es igual a 1, entonces el programa va a la línea 220; si «tecla» es 2, va a la línea 250, etc. La figura 4.3 ilustra el funcionamiento de esta instrucción.

La necesidad de dar opciones es de lo más común en un programa, y ON proporciona al programador una forma de ofrecer diversas opciones sin recurrir a gran cantidad de instrucciones IF-THEN. Observe el programa 4.7, pero no lo teclee (a menos que le guste el trabajo por el trabajo).

Programa 4.7

```
10 INPUT"Teclee el numero que elige: ",eleccion
20 IF eleccion=1 THEN GOTO 100
30 IF eleccion=2 THEN GOTO 24
40 IF eleccion=3 THEN GOTO 310
50 IF eleccion<1 OR eleccion>3 THEN 10
```

Incluso un ejemplo tan sencillo como éste demuestra que ON es mucho más rápido y cómodo. Otra aplicación de ON es la detección de errores (capítulo 6).

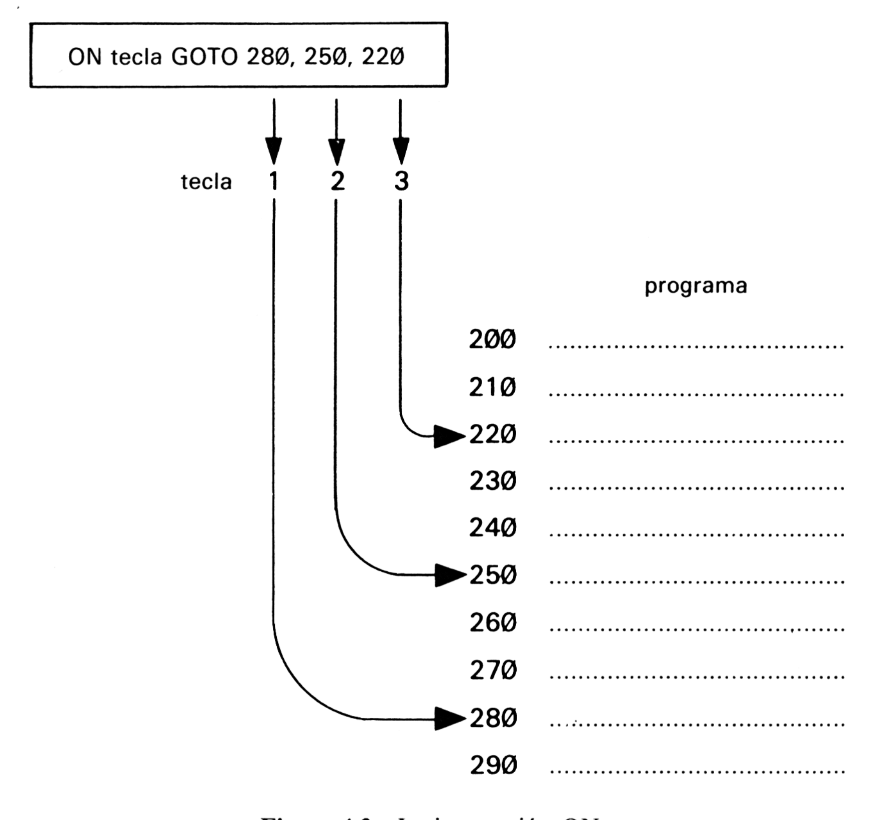


Figura 4.3 La instrucción ON.

REPETICIONES: WHILE ... WEND e INKEY$

Ya conocemos un tipo de lazo, el FOR-NEXT. Pero hay otra instrucción que se puede utilizar en situaciones en las que se quiera repetir un conjunto de acciones hasta que se cumpla cierta condición. Se trata de la instrucción WHILE ... WEND. Veamos el ejemplo del programa 4.8.

Programa 4.8

```
10 x=0
20 WHILE x<15
30 PRINT x
40 x=x+1
50 WEND
```

La línea 10 pone x a cero, y la línea 40 incrementa su valor. La línea más importante es la 20, que dice a la máquina que mientras x sea menor que

15 todas las instrucciones que siguen hasta llegar a la instrucción WEND deben ser ejecutadas. En este caso el bucle se repite hasta que x es igual a 15.

WHILE (mientras que) señala el principio del bucle. WEND da por terminado el lazo cuando se cumple la condición.

WHILE ... WEND se puede usar en muchas otras circunstancias; por ejemplo, para detectar la pulsación de cierta tecla. Para ello combinaremos WHILE ... WEND con INKEY$. Escriba el programa 4.9.

Programa 4.9

```
10 x$=INKEY$
20 WHILE x$=""
30 GOTO 10
40 WEND
50 PRINT"HA PULSADO LA TECLA ";x$
```

El bucle de las líneas 10 a 40 se repite hasta que se pulsa una tecla. La función INKEY$, como hemos explicado, detecta si se está pulsando una tecla. Cuando esto ocurre, el carácter pulsado se asigna a x$ y el programa continúa. En otro caso, la instrucción INKEY$ se ejecuta indefinidamente, pues está situada en un bucle WHILE ... WEND, lo que hace que el lazo se repita hasta que se pulse una tecla.

Estas funciones pueden ser muy útiles para determinar si se está pulsando cierta tecla, y es una buena forma de asegurar que sólo se acepte la tecla especificada. Recordaremos esto cuando hablemos de la detección de errores en el capítulo 6.

Con esto termina nuestro repaso de la manipulación de cadenas. Si usted ha entendido todo lo tratado en estos dos últimos capítulos, pronto podrá crear programas que actúen con (y no contra) el usuario.

5

Palabras, palabras, palabras

Aun con la aparición de nuevos sistemas de comunicación, área de importancia creciente en informática, la forma fundamental de comunicación sigue siendo la misma: la palabra escrita. O, para ser más exactos, la palabra sobre la pantalla, que representa la máxima comodidad desde el punto de vista de nuestra comunicación con las máquinas. Hasta ahora en este libro nos hemos centrado en la introducción de información en el ordenador y en la manipulación por el mismo de estos datos. Pero hemos trabajado únicamente con pequeñas cantidades de información sueltas, que no nos han servido de mucho. Ahora necesitamos abordar el problema de almacenar grandes cantidades de datos a los que el ordenador pueda recurrir en cualquier momento.

VECTORES

Imagine que tenemos que archivar una relación de los puestos finales de los pilotos de coches de una determinada carrera. Necesitamos el ordenador para que nos diga quién terminó en un puesto determinado. ¿Cómo vamos a introducir esta información? Digamos que tenemos que almacenar los nombres de los seis primeros corredores. Usando lo que ya hemos aprendido anteriormente, emplearíamos seis variables distintas para contener la información. Nuestro programa sería algo parecido a esto:

Programa 5.1

```
10 CLS:PRINT"Introuzca los nombres de los conduc
tores";
20 PRINT"del 1 al 6"
```

```
30 INPUT numero1$
40 INPUT numero2$
50 INPUT numero3$
60 INPUT numero4$
70 INPUT numero5$
80 INPUT numero6$
90 CLS
100 INPUT"Que puesto quiere comprobar?",comproba
r
110 ON comprobar GOTO 120,130,140,150,160,170
120 PRINT"Puesto 1: ";numero1$:GOSUB 180
130 PRINT"Puesto 2: ";numero2$:GOSUB 180
140 PRINT"Puesto 3: ";numero3$:GOSUB 180
150 PRINT"Puesto 4: ";numero4$:GOSUB 180
160 PRINT"Puesto 5: ";numero5$:GOSUB 180
170 PRINT"Puesto 6: ";numero6$:GOSUB 180
180 PRINT"Pulse barra espaciadora para continuar
"
190 a$=INKEY$:IF a$<>" " THEN 190
200 GOTO 90
```

Pero este camino no conduce a ningún sitio. En primer lugar, he definido seis variables distintas para contener los nombres de los conductores. Pero, ¿qué ocurriría si tuviera un centenar de nombres? Tendría que dedicar a cada respuesta una línea diferente. Comunicar al ordenador el número que queremos comprobar no es fácil; la única solución es preparar una contestación diferente para cada posible entrada. Bien, no necesito decirle que esto es ineficaz. Entonces, ¿qué se puede hacer para aliviar las penalidades del programador?

La respuesta está en la introducción de vectores. ¿Qué es un vector? Puede considerar que un vector es una tabla con información, algo así como tener una ficha con una serie de líneas de información escritas en ella. Véase la figura 5.1.

En la figura 5.1 sólo se ve una ficha. La ficha lleva un título que dice VENCEDORES. En cada línea hay un número de posición y un nombre. Para que el ordenador sepa que queremos teclear hasta seis nombres, comunicamos a la máquina que la tabla tiene seis elementos. Esto se hace con la instrucción DIM. Así, nuestra primera línea será:

```
10 DIM vencedor$(6)
```

Lo que hemos hecho es preparar un vector unidimensional que podrá con-

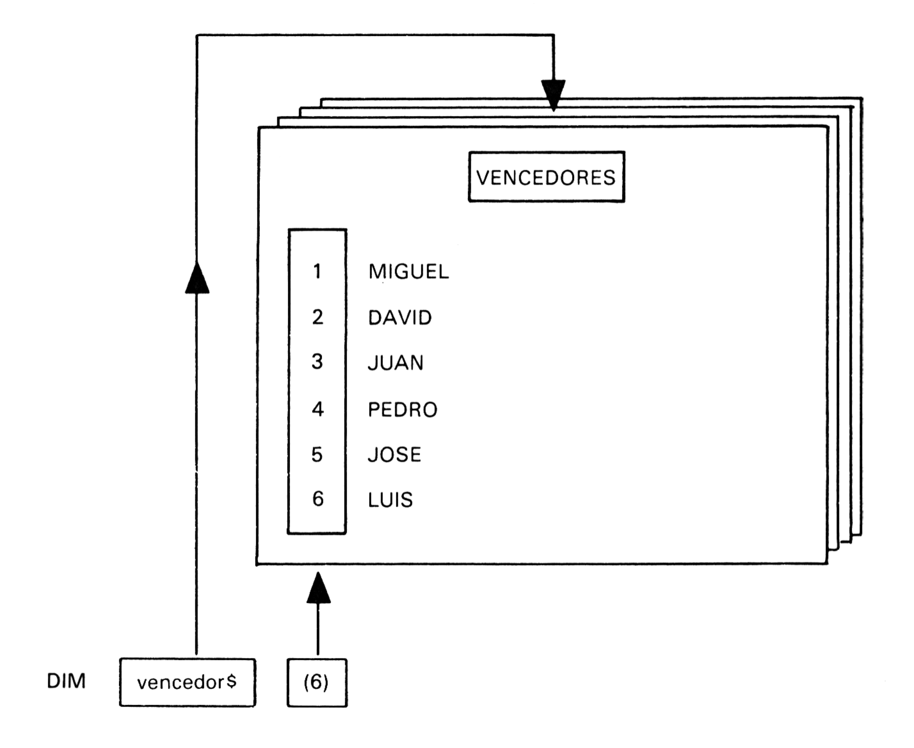


Figura 5.1 Vectores.

tener un máximo de seis datos. Ahora podemos completar el resto del programa.

Programa 5.2

```
20 PRINT"Introduzca los nombres de los conductor
es"
30 PRINT"del 1 al 6"
40 FOR entrada=1 TO 6
50 INPUT vencedor$(entrada)
60 NEXT
70 CLS
80 INPUT"Que puesto quiere comprobar ",comprobar
90 PRINT"En el puesto";comprobar;"entro "vencedo
r$(comprobar)
100 PRINT"Pulse barra espaciadora para continuar
"
110 a$=INKEY$:IF a$<>" " THEN 110
120 GOTO 70
```

Hemos introducido toda nuestra información en una única variable, vencedor$, pero en seis lugares diferentes. Mediante el bucle FOR-NEXT se va aumentando el valor de «entrada» para «apuntar» a los seis elementos del vector: vencedor$(1), vencedor$(2), etc. Además de haber ahorrado tiempo y espacio en la parte correspondiente a la introducción de los datos, también podemos ahorrar tiempo cuando recuperemos la información. La línea 90 localiza y escribe el dato correcto, ahorrando de esta forma mucho espacio. Este programa tendrá el mismo tamaño para seiscientos datos que para seis. Lo único que se habría que cambiar es la longitud del vector, y esto podemos hacerlo sin más que modificar la línea 10, es decir, DIM vencedor$(600), y el límite superior del bucle FOR-NEXT, ¡Imagínese la escritura de un programa para conjuntos de datos verdaderamente grandes utilizando el primer método!

Este último es más eficaz, pero hay situaciones en las que necesitamos introducir varios bloques de información relacionados con un suceso concreto. Volviendo a nuestro ejemplo de carreras automovilísticas, ahora queremos que también aparezca en pantalla el país a que pertenece cada conductor. Podríamos añadir la siguiente línea al programa:

```
55 INPUT "Introduzca el país", país$(entrada)
```

y cambiar las líneas 10 y 90 por las siguientes:

```
10 DIM vencedor$(6), país$(6)
90 PRINT "en el puesto"; comprobar; "entró"; venced
or$(comprobar); SPC(4);"(";país$(comprobar)")"
```

Pero a medida que añadamos más datos tendremos que aumentar el número de vectores y esto puede llegar a desbordarnos. Sería mucho más fácil si pudiésemos incorporar la información sobre el país en la misma variable que el nombre. Por supuesto, podemos hacerlo. Lo que vamos a utilizar ahora es un vector bidimensional o, si usted prefiere, una matriz o tabla de doble entrada. Observe la figura 5.2.

Cambiemos el vector de la línea 10:

```
10 DIM vencedor$(6,1)
```

De esta forma le decimos al ordenador que vencedor$ tiene doble entrada. Cambie la línea 50:

```
50 INPUT "Introduzca el nombre ", vencedor$(entrada,0)
```

DIM vencedor$

	0	1
1	MIGUEL	ESPAÑA
2	DAVID	ESPAÑA
3	JUAN	MEJICO
4	PEDRO	ARGENTINA
5	JOSE	ESPAÑA
6	LUIS	VENEZUELA

Figura 5.2 Vector bidimensional.

y añada esta otra:

```
55 INPUT "Introduzca el país ", vencedor$(entrada,1)
```

Finalmente, cambie la 90:

```
90 PRINT "En el puesto"; comprobar;"entro"; venced
or$(comprobar,0); SPC (4); "("; vencedor$(comprobar,1)")"
```

He puesto las entradas en dos líneas diferentes para mostrarle más claramente lo que está ocurriendo. En la línea 50 está vencedor$(1,0), que se iguala al nombre. En la 55 está vencedor$(1,1), donde se pone el país. El bucle FOR-NEXT incrementa el primer valor hasta que se han introducido seis nombres y los correspondientes países. La figura 5.2 resume lo que queda almacenado en cada celdilla de la variable. Pero no tenemos por qué limitarnos a matrices bidimensionales. Podemos, con la misma facilidad, definir matrices de dimensiones tales como 3×2×5×3.

Los vectores son, pues, muy adecuados para manejar grandes cantidades de datos. Para nosotros esto es potencialmente útil, ya que crearemos programas que se apoyan en el uso de una base de datos. Pero los vectores son sólo la mitad de la historia. Para que nos sean útiles necesitamos mantener los datos de forma permanente y el programa anterior solicita la introducción de los datos cada vez que lo ejecutamos. ¿Cómo podríamos almacenar datos dentro del programa? Una posibilidad sería la siguiente:

nombre1$="JEREMY"
nombre2$="PEDRO"
nombre3$="CATALINA"

pero esto no nos permitiría utilizar vectores. Una forma mejor de almacenar información es el uso de los comandos DATA y READ.

DATOS INDELEBLES: READ y DATA

Las instrucciones READ y DATA permiten guardar datos de forma permanente dentro de un programa para que luego puedan ser asignados a un vector. Veamos un ejemplo muy sencillo. El programa 5.3 almacena nombres de futbolistas y los escribe en la pantalla cuando tecleamos el número:

Programa 5.3

```
10 DIM a$(11)
20 FOR x=1 TO 11
30 READ a$(x)
40 NEXT
50 CLS
60 INPUT"Introduzca el numero del jugador ",num
70 PRINT"El jugador numero";num;"es ";a$(num)
80 a$=INKEY$:IF a$="" THEN 80
90 GOTO 50
100 END
110 DATA ARCONADA,GERARDO,CAMACHO,MACEDA,GOICOEC
HEA
120 DATA GORDILLO,SE\OR,LOZANO,SANTILLANA
130 DATA LANDABURU,LOPEZ UFARTE
```

La línea 10 dimensiona el vector. En la línea 30 nos encontramos una instrucción nueva, READ. La instrucción READ está contenida en un bucle FOR-NEXT; la primera vez que se recorre el bucle, la línea 30 intenta READ (leer) información para asignarla a la variable a$(1); el dato que encuentra es el primero de la línea 110. Toda la información está contenida en líneas que empiezan por DATA; los datos se separan mediante comas para que el ordenador sepa dónde termina un dato y empieza otro. Así pues, la instrucción READ toma el primer dato que hay en la línea; la próxima vez que se recorre el bucle lee el segundo bucle, y así sucesivamente. La figura 5.3 muestra dónde se van almacenando los datos a medida que son leídos (READ) por el ordenador.

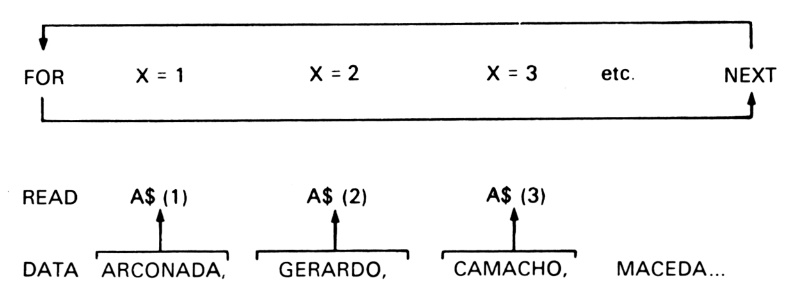


Figura 5.3 READ y DATA.

Notará que he utilizado tres líneas de instrucciones DATA. Usted puede repartir los datos en el número de líneas que quiera, siempre que ponga una instrucción DATA al comienzo de cada una de ellas. También es convencional la colocación de los datos al final del programa. Tenga en cuenta que si usa la instrucción READ en diferentes ocasiones a lo largo del programa, debe ordenar sus datos de tal forma que sean leídos por el ordenador en el orden deseado. Una cosa más, si recibe un mensaje de error DATA EXHAUSTED (datos agotados), será porque, o bien ha introducido menos datos que los que deben ser leídos, o bien pretende leer demasiados, es decir, el bucle es demasiado largo.

Ahora que entendemos las instrucciones READ y DATA, utilicémoslas para algo un poco más interesante. El objetivo final de este libro es construir programas interactivos, podemos comenzar a poner en práctica algo de lo que hemos aprendido. El programa 5.4 simula una conversación con un ordenador un tanto brusco que no está de acuerdo con usted.

Programa 5.4

```
10 MODE 2
20 DIM respuesta$(4),n$(4)
30 FOR x=0 TO 1
40 READ respuesta$(x)
50 READ n$(x)
60 NEXT
70 FOR x=0 TO 1
80 PRINT"VENGA!"
90 LINE INPUT a$
100 PRINT respuesta$(x);a$;n$(x)
110 NEXT
120 END
130 DATA "DICES QUE ",", PERO NO PUEDES DEMOSTRA
RLO"
140 DATA "QUE TONTERIA!. COMO PUEDES DECIR QUE "
," Y QUEDARTE TAN PANCHO?"
```

Ejecute el programa y, en respuesta a la cortés invitación BIEN!, teclee lo siguiente:

LOR ORDENADORES AMSTRAD SON ESTUPENDOS

y conteste a la segunda pregunta con

SON MEJORES QUE LOS SINCLAIR

Esto no es exactamente inteligencia artificial, pero nos inicia en el buen camino. La línea 100 escribe la primera parte de la respuesta, inserta la frase que usted ha introducido y finalmente añade una réplica para terminar la frase. Habrá notado que las instrucciones DATA son ligeramente diferentes en el programa 5.4; aquí las cadenas están entre comillas. Esto es debido a que, sin ellas, los espacios iniciales y finales son ignorados y es necesario dejar espacios a ambos lados de la respuesta introducida. Poniendo la frase entre comillas, todo lo que está entre ellas es utilizado, incluyendo los espacios.

Practique con este programa aumentando las instrucciones DATA y vea hasta dónde puede alargar la conversación. Por supuesto, debe limitar mucho sus respuestas, pero puede ser divertido. Más tarde, en los capítulos 8 y 9 le mostraré un interlocutor mucho más interesante.

RESTAURACIONES: RESTORE

Con frecuencia es útil poder controlar un «puntero» que señale al lugar del que queremos que se lean los datos. Algunas veces querremos modificar este puntero para leer de nuevo los datos desde el principio o desde algún otro punto concreto de las instrucciones DATA. Esto se puede conseguir con la instrucción RESTORE. Por ejemplo, se puede escribir un programa en el que la respuesta del usuario determine el lugar desde el cual han de leerse los datos.

Utilicemos el ejemplo de una tienda de comestibles. Un parroquiano quiere saber el precio de una determinada fruta o verdura. En función de la elección del cliente, el puntero de datos se coloca bien al principio de los datos sobre frutas o al principio de los datos sobre verduras. Teclee el programa 5.5:

Programa 5.5

```
10 DIM articulo$(7,1)
20 CLS
30 PRINT"De que quiere saber precios ","(F)ruta
o (V)erduras ?"
```

```
40 INPUT eleccion$
50 IF eleccion$="F" OR eleccion$="f" THEN RESTOR
E 130 ELSE RESTORE 150
60 FOR x=1 TO 7
70 FOR precio=0 TO 1
80 READ articulo$(x,precio)
90 NEXT:NEXT
100 FOR y=1 TO 7
110 PRINT articulo$(y,0);" a ";articulo$(y,1);"
pesetas"
120 NEXT
130 DATA MELOCOTONES,150,MANZANAS,100,NARANJAS,1
20,UVAS,55
140 DATA CEREZAS,60,MNADARINAS,80,FRESAS,250
150 DATA PATATAS,45,TOMATES,70,GUISANTES,25,SANA
HORIAS,77
160 DATA CEBOLLAS,40,REPOLLO,65,LECHUGAS,50
```

En la línea 50 es donde se decide qué datos se van a utilizar. Si el cliente se ha interesado por la fruta, el puntero de datos se coloca apuntando a la línea 130; y en caso contrario, a la 150, que es donde se encuentran los datos sobre verduras. Por tanto, RESTORE dice al ordenador a partir de qué lugar debe leer los datos.

Finalmente, voy a mostrar una instrucción más. Se trata de RND, que nos permite elegir un suceso aleatoriamente. ¡Al mismo tiempo voy a desvelar un secreto: voy a enseñarle cómo escriben los escritores sus libros en la época actual! ¿Cómo? ¡Haciendo que se los escriba el ordenador!

UN ESCRITOR: RND

Todo lo dicho sobre los datos indica el camino obvio para escribir textos. Hacer que el ordenador genere aleatoriamente texto para usted. Esto es muy fácil; la mayor dificultad radica en la elección de las frases a utilizar. El siguiente programa usa cuatro variables vectoriales. He reservado espacio para cuatro elementos en la instrucción DIM, pero sólo he utilizado la mitad con mis datos. Vea si puede añadir frases, o si prefiere (¡lo que seguramente ocurrirá!) cambiarlas. En esta situación es de gran ayuda el perfecto dominio de un lenguaje «humano».

El programa 5.6 lee las frases, las asigna a los vectores y a continuación genera una frase escogiendo en cada caso una de las dos posibles opciones para cada una de las cuatro partes de la frase. He utilizado la función RND para escoger aleatoriamente uno u otro de los números 0 y 1. Si añaden

más datos, el número aleatorio tendrá que poder ser mayor. La función RND es muy simple, como demuestran las líneas 130 a 160. Si desea elegir un número aleatorio comprendido entre 1 y 125, necesitará una instrucción como ésta:

variable = INT(RND*125)

El número entre paréntesis es el más grande que se puede obtener. Si necesita un número entero, use INT. Esta instrucción redondea el número al entero más próximo. Escriba el programa 5.6 y observe.

Programa 5.6

```
10 CLS
20 DIM frase1$(4),frase2$(4),frase3$(4),frase4$(
4)
30 GOSUB 60
40 GOSUB 100
50 END
60 FOR x=0 TO 1
70 READ frase1$(x),frase2$(x),frase3$(x),frase4$
(x)
80 NEXT
90 RETURN
100 GOSUB 130
110 PRINT frase1$(x1);" ";frase2$(x2);" ";frase3
$(x3);" ";frase4$(x4)
120 RETURN
130 x1=RND(1)
140 x2=RND(1)
150 x3=RND(1)
160 x4=RND(1)
170 RETURN
180 DATA En este mundo moderno, la humanidad se
ha puesto al borde de la autodestruccion
190 DATA y vamos a llegar a
200 DATA limites insospechados
210 DATA Desde la llegada de los ordenadores
220 DATA la vida se ha vuelto insoportable
230 DATA y las normas de conducta estan en,un ab
ismo sin fin
```

Este programa es una sencilla muestra de lo que se puede conseguir reagrupando frases de forma aleatoria. Por ejemplo, cuando ejecute el programa puede obtener frases como las siguientes:

> En este mundo moderno la vida se ha vuelto insoportable y vamos a llegar a un abismo sin fin.

> Desde la llegada de los ordenadores la humanidad se ha puesto al borde de la autodestrucción y las normas de conducta están en límites insospechados.

Bueno, no nos pongamos tan tristes. Es interesante preparar partes diversas de una frase y reunirlas en algo que suene gramaticalmente correcto, aunque sea una tontería. Pensando cuidadosamente, usted puede ser capaz de hacer que su Amstrad le redacte informes, ensayos y proyectos. Bueno, es posible.

Con esto concluye por el momento nuestro estudio de los vectores y los datos, aunque volveremos a hablar de ellos más tarde, cuando los usemos para cosas más grandes y mejores. Las instrucciones DIM, READ y DATA constituyen el núcleo de los programas que simulan tener alguna inteligencia. En todo caso, nos serán muy útiles para almacenar grandes cantidades de información.

6

¡Siempre lo inesperado!

Cuando se escriben programas para uno mismo, muchas veces es más sencillo tomar atajos. No hay inconveniente si usted va a ser el único usuario de los programas; pero muchas veces su software (sus programas) será utilizado por otras personas, y ahí es donde surge el problema. A menos que usted sea capaz de leer la mente o tenga algún otro poder extrasensorial, la gente que utilice su programa no vacilará en hacer lo imposible para confundir a su obra maestra. Pueden ignorar sus instrucciones y teclear algo fuera del margen de datos que permite el programa, o pulsar accidentalmente alguna tecla equivocada. Desde luego, no es fácil que prevea todas las posibilidades, pero en la medida de lo posible intente minimizar las consecuencias del error humano. A eso está dedicado este capítulo: a detectar lo inesperado.

Hasta ahora, en los programas que hemos escrito, no hemos pensado en detectar los errores que puede cometer el usuario. ¿Qué tipo de cosas pueden ocurrir? ¿Que se pulse la tecla ESC accidentalmente? ¿Que se introduzca información equivocada? Estos son riesgos muy corrientes en los programas. Es esencial tratar de cubrir en lo posible estos resquicios en su programación. Comencemos con una frase muy utilizada en el lenguaje comercial de ordenadores: facilidad de uso.

FACILIDAD DE USO: PRESENTACIÓN EN PANTALLA, DETECCIÓN DE ERRORES Y ENTRADAS RIGUROSAS

¿Cuántas veces ha oído la frase anterior? ¿Cuántos programas de los que ha comprado le han dejado confundido a la primera ocasión? Quizás los programas son fáciles de usar, pero para quien los escribió. El primer sitio en que un programa puede causar confusión es en sus instrucciones. Si es ne-

cesario, proporcione instrucciones concisas y claras al principio del programa. Si quiere que un usuario interactúe con el ordenador, no puede esperar que adivine lo que debe hacer. Póngale las cosas fáciles al usuario.

Además, los mensajes que emita a lo largo del programa también son importantes. Reconozco que en el capítulo anterior yo no he seguido mis propios consejos. En varias ocasiones a lo largo del libro, mientras nos concentramos en un determinado aspecto, y para no cansar más sus agotados dedos, he tomado el camino más corto. La consecuencia es frecuentemente que cuando se pone en marcha un programa lo único que ve es un signo de interrogación. Si esto le ha resultado frustrante, entenderá fácilmente lo que estoy tratando de explicar. Tome como ejemplo el programa 6.1.

Voy a escribir un pequeño programa que representa el proceso que usted debe seguir para sacar dinero de un cajero automático situado en el exterior de su banco. En el programa usted debe introducir un número de cuenta de seis dígitos y su contraseña de cuatro letras. Después tecleará la cantidad que solicita. Que lo reciba o no depende de la cantidad que tenga en su cuenta; no puede sacar más de veinte mil pesetas en cada operación. Escribamos el programa.

Programa 6.1

```
10 CLS
20 PRINT"INTRDUCCION DE DATOS"
30 INPUT num
40 IF num=123456 THEN GOSUB 50 ELSE 30
50 INPUT contra$
60 IF contra$="USUARIO" THEN GOSUB 80 ELSE 50
70 RETURN
80 INPUT"Teclee cantidad de dinero: ",dinero
90 GOSUB 110
100 RETURN
110 credito=17000
120 IF dinero<20000 AND dinero<credito THEN GOSU
B 150 ELSE 130
130 PRINT"Lo siento. No es posible el pago.":END
140 RETURN
150 PRINT"Dinero entregado:";dinero
160 RETURN
```

Al comenzar la ejecución del programa usted se encuentra ante un signo de interrogación que le solicita la introducción de ciertos datos. ¿Qué debe teclear? ¿Un número? ¿Su contraseña? ¿La cantidad de dinero que quiere? Si consulta el listado sabrá que el primer dato solicitado es el número de cuenta, y a continuación la contraseña.

Mejoremos los mensajes cambiando las líneas 20 y 50:

```
20 PRINT"Teclee el numero de su cuenta"
50 INPUT"Introduzca su contrase\a personal"
```

Así está mejor. Pero podemos hacer algo más para ayudar al usuario. ¿Qué ocurre si éste comete un error? Tendrá que volver a teclear su entrada, pero tal vez no sepa qué equivocación ha cometido, así que deberíamos decirle dónde ha errado para que la próxima vez todo vaya felizmente. Los cambios son los siguientes:

```
40 IF num=123456 THEN GOSUB 50 ELSE GOSUB 170
60 IF contra$="USUARIO" THEN GOSUB 80 ELSE GOSUB
 230
```

y agregue las líneas 170 a 270:

```
170 PRINT"El numero de cuenta no es correcto"
180 PRINT"Recuerde que debe teclear"
190 PRINT"seis digitos"
200 FOR x=0 TO 2000:NEXT
210 GOTO 10
220 RETURN
230 PRINT"La contrase\a no es correcta"
240 PRINT"Por favor, intentelo otra vez"
250 FOR x=0 TO 2000:NEXT
260 GOTO 50
270 RETURN
```

Ahora tenemos preparados mensajes adicionales por si el usuario comete un error. Observe que he usado una instrucción FOR-NEXT para realizar una breve pausa antes de volver a solicitar la entrada. En algunas circunstancias puede ser preferible obligar al usuario a pulsar una tecla para volver al principio. También se puede asear un poco la pantalla borrándola cada vez que se efectúa una entrada, de forma que si el usuario comete unos cuantos errores, no queda en ella una lista de mensajes de error. Una vez que se sienta satisfecho en este aspecto pase al siguiente problema.

El problema radica en que, cuando el pago es aprobado, el programa también escribe el mensaje de que no se puede hacer el pago. Esto se puede resolver haciendo que el programa termine cuando se efectúa el pago, pero

la verdad es que el programa está mal estructurado. Examine el programa 6.2, que ha sido estructurado con subrutinas. La presentación en pantalla es mucho mejor cuando se comete alguna equivocación:

Programa 6.2

```
10 CLS
20 GOSUB 70
30 GOSUB 120
40 GOSUB 170
50 GOSUB 250
60 END
70 CLS
80 PRINT"Teclee el numero de su cuenta"
90 INPUT num
100 IF num=123456 THEN 110 ELSE 310
110 RETURN
120 CLS
130 PRINT"Introduzca su contrase\a personal"
140 INPUT contra$
150 IF contra$="USUARIO" OR contra$="usuario" TH
EN 160 ELSE GOSUB 370
160 RETURN
170 CLS
180 PRINT"Teclee cantidad de dinero: "
190 INPUT dinero
200 PRINT"Ha solicitado";dinero;"pesetas. Es cor
recto?"
210 INPUT resp$
220 IF resp$="S" OR resp$="s" THEN GOSUB 250 ELS
E 230
230 IF resp$="N" OR resp$="n" THEN GOTO 170 ELSE
 200
240 RETURN
250 credito=17000
260 IF dinero<20000 AND dinero<credito THEN PRIN
T"De acuerdo. Recoja las";dinero;"pesetas.":END
270 IF dinero>20000 THEN PRINT"Supera el limite
autorizado":END
280 IF dinero>credito THEN PRINT"No tiene fondos
 suficientes":END
290 RETURN
310 PRINT"El numero de cuenta no es correcto"
320 PRINT"Recuerde que debe teclear"
```

```
330 PRINT"seis digitos"
340 FOR x=0 TO 2000:NEXT
350 GOSUB 70
360 RETURN
370 PRINT"La contrase\a no es correcta"
380 PRINT"Por favor, intentelo otra vez"
390 FOR x=0 TO 2000:NEXT
400 GOSUB 120
410 RETURN
```

Observe que el programa analiza con rigor las respuestas para que sean correctas. El nivel de rigor que incorpore a sus programas es cosa suya y puede depender de la naturaleza del programa. Si recuerda, al final del capítulo 3 he dicho que el programa ASCII (programa 3.10) contiene algunas imperfecciones. ¿Las descubrió ya?

El problema está en la subrutina que transforma un valor ASCII en un carácter, pues no hay un detector que elimine valores incorrectos. Por ejemplo, pulsando el valor 7 el código se convierte en un pitido. Lo que necesitamos es una comprobación del número que se introduce. Es muy sencillo. Añadiendo las siguientes líneas, el programa aceptará únicamente los valores permisibles. (En este caso he limitado los valores a los comprendidos entre 32 y 126, que abarcan el conjunto completo de caracteres, aunque no los definibles por el usuario.) Cargue el programa 3.10 y añádale las siguientes líneas:

Programa 6.3

```
255 IF codigo<32 OR codigo>126 THEN GOSUB 300
300 PRINT"El codigo introducido està fuera"
310 PRINT"del margen correcto."
320 a$=INKEY$:IF INKEY$=" " THEN 330 ELSE 320
330 GOSUB 50
340 RETURN
```

Cuando ejecute el nuevo programa verá que éste impide la introducción de números ajenos al margen correcto. Hay que añadir una cosa más al programa para protegerlo contra dedos despistados. Podemos hacer que si se pulsa accidentalmente la tecla ESC, el programa se reinicie a partir de un cierto punto. Para ello necesitamos dos instrucciones. Una ya la hemos encontrado anteriormente: ON. La otra es BREAK, que permite asignar una función al hecho de pulsar la tecla ESC.

GESTION DE ERRORES: ON BREAK, ON ERROR, ERR, ERL, KEY Y KEY DEF

Además de en la toma de decisiones, como en el programa menú, la instrucción ON sirve también para detectar cuándo se está pulsando la tecla ESC durante la ejecución del programa. Si se pulsa la tecla ESC, lo normal es que el programa se detenga. Pero mediante la instrucción ON BREAK podemos decir al ordenador qué acción debe emprender en tal caso. Por ejemplo, si ejecuta el programa 3.10 (en su versión completa), el programa se detiene al pulsar ESC. Para impedirlo añada esta línea:

```
5 ON BREAK GOSUB 50
```

Lo que acaba de decirle al ordenador es que, si se pulsa la tecla ESC, el programa debe volver a empezar, lo que significa que volverá al menú principal. Por supuesto, también se podría especificar que la acción fuera cualquier otra; todo depende de lo que se ponga a continuación de ON BREAK.

Tome buena nota, sin embargo, de que si incluye este detector de error, debe hacerlo cuando haya terminado ya el programa, pues de otra forma no será capaz de detenerlo y lo perderá, pues tendrá que apagar la máquina. La única forma de evitar esta situación es tener una instrucción ON BREAK STOP dentro del programa por si se pulsa dos veces la tecla ESC. El otro tipo de instrucción ON es ON ERROR. Cuando se detecta un error en el programa, éste es enviado a una determinada línea dentro; por ejemplo, ON ERROR GOTO 60.

Recuerde que esta instrucción debe ser la última que incorpore al programa o le será difícil depurarlo. Es muy fácil olvidar que se ha puesto una instrucción ON ERROR en un programa y gastar tiempo ejecutándolo una y otra vez y sufriendo los mismos fallos sin ver qué es lo que ocurre.

Otras instrucciones muy útiles son las que nos permiten redefinir la función de una tecla. Hay dos instrucciones de este tipo. KEY redefine una nueva función para una tecla dando al usuario la oportunidad de conseguir acciones complejas por la simple pulsación de la misma. Así, una definición podría ser:

```
KEY 140, "NEW" + CHR$(13)
```

Esto indica a la máquina que cuando se pulsen simultáneamente las teclas CONTROL y ENTER, debe actuar como si se hubiera tecleado NEW y se hubiera pulsado ENTER. (Consulte la información que se da a este respecto en el manual del usuario.) También podemos cambiar cualquier tecla del teclado para que genere otro carácter. Por ejemplo, podemos hacer que al

pulsar la tecla Q aparezca una A en la pantalla. La instrucción es KEY DEF; se usa de la siguiente forma:

KEY DEF 67, 1, 65

El primer número es el de la tecla que se va a redefinir (vea el manual); el segundo indica si la tecla ha de ser repetitiva. El 1 activa la repetición; el 0 la desactiva. El tercer número es el código ASCII del carácter. Así, en nuestro ejemplo ponemos el 65 para especificar la letra A. Añadiendo dos parámetros más, podremos utilizar también las teclas SHIFT y CONTROL en conjunción con cualquier otra.

De esta forma se pueden asignar funciones a las distintas teclas y hacer más fácil al usuario la introducción de respuestas. Por ejemplo, podría asignar a algunas teclas las funciones de contestar SI o NO, NORTE, SUR, etc. Esto hace el programa más atractivo y desde luego más fácil de usar.

Finalmente, podemos decir al usuario qué error se ha cometido utilizando las instrucciones ERR y ERL. ERR es una variable cuyo valor es el número del error. (Los números de error están relacionados en el manual del usuario.) ERL da el número de la línea donde se ha producido el error. Ambas variables son muy útiles para el tratamiento de los errores.

La detección de errores es importante en todos los programas y hace la vida mucho más fácil para el usuario. A fin de cuentas, es responsabilidad del programador garantizar que el tratamiento de los errores sea bueno; y el que sea más o menos bueno depende de su habilidad y del cuidado que ponga al hacer el programa. Suponga siempre que la persona que va a utilizar el programa sabe muy poco, quizá nada.

En el siguiente capítulo veremos alguna aplicación adicional de la detección de errores. Como he mencionado antes, en algunos de los programas de este libro he omitido voluntariamente las trampas para errores. El lector puede dedicar unos minutos a tratar de mejorarlos. Utilice diagramas de flujo para planificar las acciones de su programa y para determinar qué curso debería tomar cuando se produce un error. Esto puede resultar algo laborioso, pero se verá recompensado con una ejecución sin problemas.

7

Manejo de textos

En los capítulos precedentes hemos visto cómo utilizar las diversas instrucciones del BASIC del Amstrad para controlar la entrada de información en el ordenador. En este capítulo vamos a ir un poco más lejos y a aprender técnicas para hacer sus programas más atractivos mediante la inserción de rutinas que escriban textos en pantalla de distintas maneras. Se trata de ofrecer alternativas al simple hecho de escribir en la pantalla los mensajes amontonados. Los gustos personales juegan un importante papel en la forma de presentar una pantalla llena de información, pero lo cierto es que un programa puede causar mejor o peor impresión según lo cuidados que estén estos detalles.

Permítame ponerle un ejemplo. Al programa 7.1 le llamo mi alarma contra intrusos. El programa deja un mensaje escrito en la pantalla, advirtiendo a cualquiera que se acerque para que no toque el teclado. Pero, siendo los humanos curiosos por naturaleza, es casi seguro que alguno tocará una tecla. Y entonces es cuando podemos divertirnos un poco y poner en práctica alguna de las ideas que hemos aprendido. No le digo más por el momento; lo mejor es que escriba el programa. Conserve los mismos números de línea que aparecen aquí, porque añadiremos algunas cosas más adelante:

Programa 7.1

```
10 ZONE 40
20 MODE 0
30 PAPER 5:BORDER 2:PEN 10:CLS:PRINT,,,
40 PRINT SPC(6);"NO TOQUE"
50 PRINT,,,:PRINT SPC(5);"EL TECLADO"
60 a$=INKEY$:IF a$<>"" THEN 70 ELSE 60
70 CLS
80 MODE 1:PAPER 4:BORDER 7:PEN 10:CLS
```

```
90 unafrase$="YA TE LO ADVERTI. LA HAS HECHO BUE
NA!!!!"
100 PRINT unafrase$
110 ENV 1,4,3,1,1,0,19,7,-120,4
120 SOUND 1,100,7,14,1
130 a$=INKEY$:IF a$<>"" THEN 140 ELSE 120
140 MODE 0
150 INK 0,4,7:CLS:PRINT,,,,
160 PRINT "DEMASIADO TARDE!!!!"
170 WHILE x=0
180 ENV 1,4,3,1,1,0,19,7,-120,4
190 SOUND 1,100,7,14,1
200 WEND
```

Ahora ejecute el programa y pulse una tecla cualquiera. Verá aparecer un mensaje de error y además sonará la alarma. Puede detener el programa pulsando ESC. Para completar el tratamiento del error, puede incluir la instrucción ON BREAK, pero recuerde grabar antes el programa, pues no podrá interrumpirlo después de añadir dicha instrucción. Si usted presiona otra tecla, en el monitor aparecerá otro mensaje informándole de que llega demasiado tarde. Observe cómo he interceptado los errores para dirigir el programa a diferentes líneas. Usted puede mejorarlo utilizando los métodos de gestión de errores estudiados en el capítulo anterior. La línea 30 fija los colores de papel, margen y pluma; usted puede cambiarlos en función del monitor que tenga y de los efectos que más le gusten. La línea 60 espera hasta que se toque el teclado; cuando esto sucede, el programa nos presenta la siguiente página. El lazo WHILE que hay al final del programa genera un sonido de duración indefinida, puesto que x siempre vale cero.

La línea 110 es la clave de la alarma, pues define las características del ruido por medio de la instrucción ENV; la línea 120 produce el sonido. Las demás instrucciones del programa deberían serle familiares. Usted puede disfrutar modificando el programa para ejecutarlo en diversos modos de pantalla y crear diferentes efectos en el monitor. La figura 7.1 nos muestra un diagrama de flujo de este programa.

El programa realiza su tarea bastante bien, pero no hay suspense en la presentación del mensaje. ¿Suspense? ¡Sí, emoción! Podemos darle vida al programa haciendo que las dos frases aparezcan lentamente en la pantalla, lo que las hará más interesantes. Para ello tenemos que aprender a escribir textos en la pantalla como si se tratase de un teletipo lento.

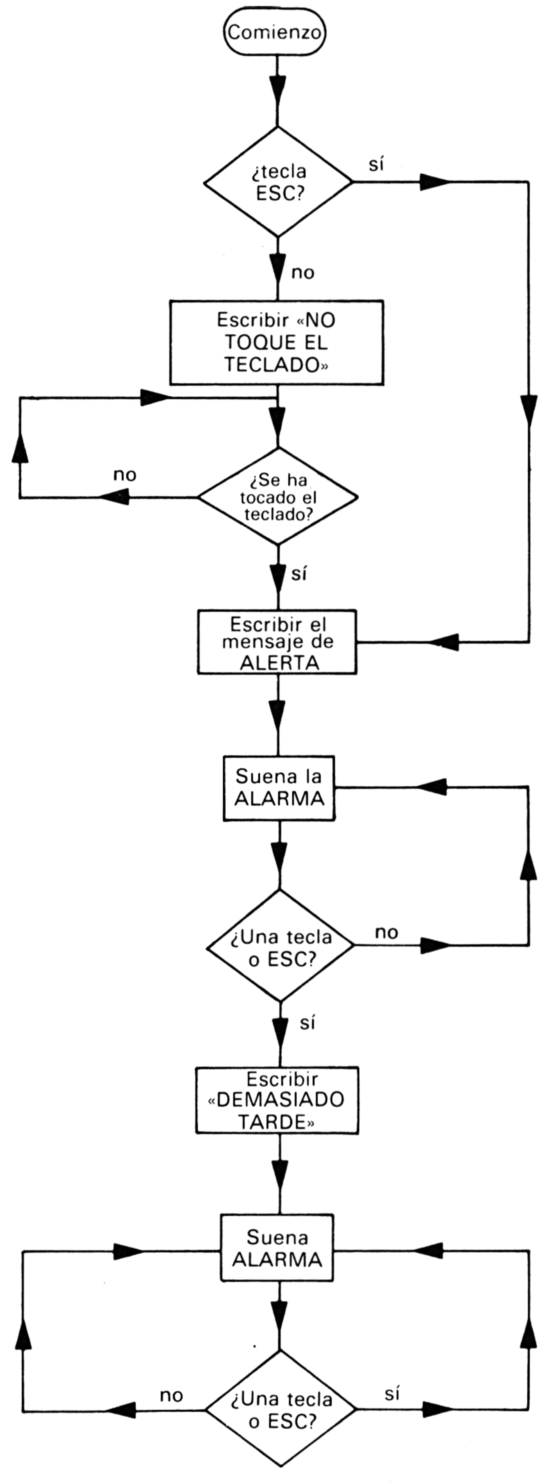


Figura 7.1 Alarma contra intrusos.

TELETIPO: UNA RUTINA QUE SIMULA A UN TELETIPO LENTO

Ya hemos utilizado instrucciones tales como MID$ para escribir, de forma selectiva, caracteres de una cadena. Podemos aplicar estas ideas a la elaboración de un pequeño programa que escriba una frase lentamente, letra a letra. Para empezar, teclee el programa 7.2:

Programa 7.2

```
5 ZONE 40
10 CLS
20 unafrase$="YA TE LO ADVERTI. LA HAS HECHO BUE
NA!!!!"
30 PRINT,:FOR x%=1 TO LEN(unafrase$):PRINT MID$(
unafrase$,x%,1);
40 GOSUB 50:NEXT:PRINT:END
50 FOR a%=150 TO 0 STEP -1:NEXT:RETURN
```

Ejecute el programa y verá que el contenido de unafrase$ se escribe lentamente en la pantalla. La idea es muy sencilla. En primer lugar, la línea 30 establece un bucle que escribe una letra de la frase cada vez que se ejecuta. Pero esto no basta, porque la máquina trabaja demasiado deprisa como para que el efecto sea observable. Tenemos que insertar también un bucle que ocasione un retardo para que la aparición en la pantalla sea más lenta. El GOSUB 50 de la línea 40 llama a la subrutina retardadora antes de que se escriba la letra siguiente. La subrutina es simplemente un bucle FOR-NEXT que se ejecuta sin hacer nada.

Podríamos incorporar este mecanismo al programa 7.1 para conseguir un efecto mucho más teatral. Haga los siguientes cambios en el programa: Cambie la línea 100:

```
100 PRINT,: FOR x% = 1 TO LEN(unafrase$): PRINT MID$
(unafrase$,x%,1);
```

y añada las líneas siguientes:

```
105 GOSUB 107: NEXT: PRINT
107 FOR a% = 150 TO 0 STEP -1: NEXT: RETURN
```

Ejecute este programa. Creo que estará de acuerdo en que hemos conseguido un efecto más dramático haciendo más lenta la aparición del texto. Es una cuestión de gusto personal el que usted use o no este tipo de presenta-

ción en sus programas, pero la rutina de teletipo lento da la impresión de que la máquina le está respondiendo.

Trate de adaptar alguno de los programas de este libro usando esta idea del teletipo. El método que usted emplee para presentar el texto puede realzar o arruinar un programa; siempre es interesante experimentar con diferentes efectos. Las instrucciones de manejo de cadenas que vimos en el capítulo 3 son particularmente útiles para conseguir nuevos efectos. Además, lo que importa es que haga llegar su mensaje al usuario de forma efectiva. Después de todo, en esto consiste la interacción.

8

Sigmund: un programa interactivo

Hemos visto la forma de usar una amplia gama de instrucciones para manipular un texto introducido por el teclado, pero aún no hemos escrito un programa que parezca inteligente. Bien, ha llegado el momento de escribir un gran programa que interactúe plenamente con el usuario. En los dos capítulos siguientes escribiremos Sigmund, que es un psiquiatra electrónico (¡premio para quien adivine en quién estaba pensando cuando puse nombre al programa!). En este capítulo estudiaremos la planificación de este programa y las técnicas empleadas. El programa está en el capítulo 9, pero no lo teclee todavía. Lea antes este capítulo entero para comprender los principios en que se basa el programa, para que cuando finalmente lo escriba entienda mucho mejor lo que hace.

Lo primero que haré es darle algunos antecedentes del programa. La idea original de Sigmund está basada en uno de los primeros programas de inteligencia artificial: Eliza. Ha habido muchos programas de este estilo y hoy día continúan las investigaciones para que los ordenadores sean «inteligentes». Programas como Eliza estaban escritos para grandes ordenadores con el fin de simular una conversación entre paciente y médico. Eliza engañó a muchas personas que lo utilizaron y quedaron convencidas de que el ordenador comprendía realmente sus problemas. Como verá en el programa Sigmund, esto no es más que un farol, pero extremadamente efectivo.

¿Cuál es el objetivo del programa? Sigmund está pensado para simular una conversación con un psicoanalista, pero no un psicoanalista cualquiera. La idea subyacente al programa es conseguir que el usuario piense que realmente está conversando con alguien, y que su conversación está siendo escuchada y seguida de forma inteligente. Se trata de un proyecto muy ambicioso, pero hemos aprendido muchas cosas que podemos reunir en un programa de esta naturaleza. ¿Por dónde se empieza un proyecto como éste?

Usted ya ha visto que permitir al usuario introducir cualquier cosa que se le ocurra puede causar problemas. No podemos introducir en el ordenador el diccionario de la Real Academia entero, ni un conjunto completo de reglas gramaticales (¡suponiendo que las entendiéramos!) que nos permitan cubrir todas las intrincadas sutilezas del idioma. Lo que queremos conseguir es una forma de simular contestaciones aceptables para el usuario sin recurrir a escribir todas las posibles respuestas, lo que constituye una tarea imposible, a menos que usted haya dado con la solución. Lo que podemos hacer es dar por supuestos ciertos hechos sobre la forma en que los humanos se comportan y piensan.

Como seres humanos, tenemos una enorme imaginación. Éste es el origen de nuestra inventiva. La habilidad de imaginar posibles sucesos y extraer ideas puede llevarnos a captar en una determinada situación cosas que no son totalmente evidentes. Todos lo hemos hecho alguna vez, y es precisamente en esto en lo que Sigmund confía para engañar al usuario. Lo que supone es que un usuario humano entenderá en sus respuestas más de lo que literalmente se dice. No es una idea tan ingenua como puede sonar en principio.

Piense en una conversación con un amigo. Durante parte de ella usted estará escuchando y hará comentarios ocasionales para demostrar que está atento y ha entendido lo que se le ha dicho. Cuando usted va a un analista, como cliente que es, hablará mucho y el analista le animará a hablar mediante comentarios incidentales. Lo que podríamos hacer, pues, es tener una serie de frases que el ordenador pueda soltar aleatoriamente sin tener en cuenta lo que usted diga. Pero este truco no nos serviría durante mucho rato. Por ejemplo, imagine que tenemos una reserva de cuatro frases para replicar al usuario. Estas frases podrían ser las siguientes:

CUENTAME MAS COSAS
HUMM. MUY INTERESANTE
CONTINUA
ENTIENDO

Con estas cuatro contestaciones y la función RND, podríamos dejar que el usuario teclee lo que desee y darle una respuesta elegida aleatoriamente, y luego repetir todo el proceso. La figura 8.1 muestra un posible diagrama de flujo para un programa que realice esa función.

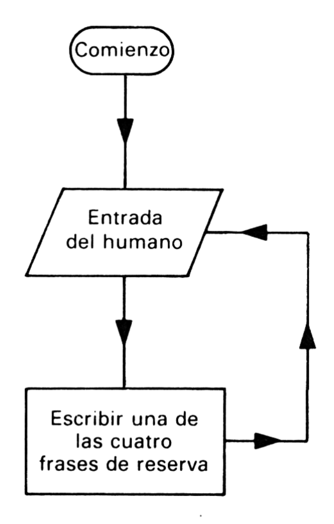


Figura 8.1 La función RND.

Pero esto no es nada convincente. Imagine la conversación que resultaría:

HUMANO:	TENGO SUEÑOS TERRRIBLES
ORDENADOR:	CUENTAME MAS COSAS
HUMANO:	SUEÑO CON GAVIOTAS
ORDENADOR:	HUMM. MUY INTERESANTE
HUMANO:	Y ESTOY MUY DEPRIMIDO POR ESO
ORDENADOR:	ENTIENDO
HUMAMO:	¿CREES QUE ESO ES ANORMAL?
ORDENADOR:	CONTINUA

El ordenador no sería capaz de mantener una conversación durante mucho tiempo usando únicamente frases en conserva. De hecho, el programa fracasa en cuanto se le hace una pregunta al ordenador. Si tecleamos las frases anteriores en el programa Sigmund, las contestaciones serán muy diferentes. Observe el diálogo:

HUMANO:	TENGO SUEÑOS TERRIBLES
ORDENADOR:	¿APARECEN OTRAS PERSONAS EN ELLOS?
HUMANO:	NO, SUEÑO CON GAVIOTAS
ORDENADOR:	¿CON QUE FRECUENCIA SUEÑAS CON GAVIOTAS?
HUMANO:	A MENUDO ¿CUAL SERA LA CAUSA?
ORDENADOR:	LA CAUSA SIEMPRE ES DIFICIL DE ENCONTRAR

Parece como si las frases introducidas hubieran sido completamente comprendidas; las respuestas parecen inteligentes. Pero, por supuesto, sabemos que el secreto tiene que estar en algún lugar del programa. Lo que he hecho es ir un poco más allá de la idea de las frases en conserva. El ordenador está utilizando todavía respuestas preparadas, pero con una mayor libertad en la forma de construir las respuestas. Dependen de lo que el usuario teclea.

COINCIDENCIAS CON PALABRAS CLAVE

Sigmund utiliza el método de buscar una palabra conocida en la frase escrita por el usuario. En otras palabras, busca una palabra clave mediante instrucciones de manejo de texto tales como INSTR. Lo hace en un orden predeterminado: busca primero un verbo o un adverbio; si no encuentra ninguno (en una lista programada en líneas de DATA), busca un sujeto, es decir YO, TU, ELLOS, etc. Si no encuentra nada de esto en la frase, echa mano de una reserva de respuestas como las que vimos antes, pero sólo como último recurso.

Por tanto, el ordenador examina una serie de posibilidades y decide la acción a tomar. La técnica consiste en almacenar una colección de respuestas, algunas parciales y otras completas. Fíjese en la instrucción DATA que sigue:

```
DATA SUEÑO, CON QUE FRECUENCIA SUEÑAS%,
PUEDES RECORDAR TUS SUEÑOS?, TE ENVIDIO ... YO
NO PUEDO SOÑAR
```

Es el aspecto típico de los datos que hay en Sigmund y contienen una contestación completa o partes de ella. La primera palabra es la *clave*, que se asigna a un vector bidimensional; las tres frases restantes son las contestaciones asociadas con dicha clave. El signo % indica al ordenador que debe añadir la parte de la frase introducida por el usuario que sigue a la palabra clave. Por ejemplo, si el usuario escribe:

SUEÑO CON BARCOS

la respuesta puede ser:

¿CON QUE FRECUENCIA SUEÑAS CON BARCOS?

El resto de la frase del usuario, CON BARCOS, se toma a partir del punto en que termina la palabra clave y se la coloca en la posición en que estaba el signo %. Mediante INSTR se examina la respuesta. Si aparece el símbolo

%, se añade el resto mencionado. Así, si examina el listado de Sigmund verá que la subrutina GOSUB 400 contiene la línea:

```
mira = INSTR(respuesta$,"%")
```

La subrutina GOSUB 430 escribe la respuesta adecuada, incluido el resto de la frase del usuario.

Como los sujetos son las partes de la oración más frecuentes, no se los busca hasta el final, cuando no se ha encontrado otra palabra clave. Su efecto es simplemente invertir la frase. Por ejemplo, si el usuario teclea:

YO TENGO UN RAMO DE FLORES

el ordenador contestará con:

TU TIENES UN RAMO DE FLORES

dando la impresión de que está confirmando lo que el humano ha dicho. Lo que ocurre es que encuentra la palabra clave YO TENGO y coloca en su lugar la frase TU TIENES más el resto de la frase tecleada, que se añade al final.

Por último, si no encuentra ninguna palabra clave, responde con una de las frases de reserva. El programa continúa de forma incansable escribiendo contestaciones que son apropiadas para muchas situaciones en que se usa una palabra clave. Hay veces en que el resultado no es el apetecido, pero funciona en un buen porcentaje de ocasiones. Se podrían introducir mejoras en el programa, pero las aplazaremos hasta cuando haya tecleado Sigmund.

Una meditación final. ¿Indican los programas como Sigmund que el ordenador y el programa tienen cierta inteligencia, o simplemente nos muestran cómo podemos sacar partido de la credulidad humana?

9

Sigmund: el programa

Después de tanto esperar, hemos llegado por fin a nuestro primer programa verdaderamente comunicativo. Sigmund es imprevisible. A veces es cariñoso y atento; en otras ocasiones, insultante. Casi siempre sentirá que es humano, pero hay momentos en que el ordenador que hay dentro se mostrará bruscamente como es. A diferencia de otros programas de este estilo que se han publicado anteriormente, he hecho que las respuestas de Sigmund sean variadas, de forma que no se pueda predecir fácilmente el tipo de contestación que puede dar. Teclee[1] el programa cuidadosamente, ya que es muy largo, y use la función AUTO para numerar las líneas, que he mantenido uniformemente espaciadas. Después del listado encontrará nuevos detalles y comentarios.

Programa SIGMUND (inglés)

```
10 MODE 2
20 ZONE 80
30 PRINT"HELLO MY NAME IS SIGMUND.","WHAT IS YOU
RS?"
40 INPUT name$
50 PRINT,,,"NICE TO MEET YOU, ";name$
60 PRINT,"TELL ME WHY YOU WANT TO TALK TO ME"
```

[1] A continuación damos dos versiones del programa, una en castellano y otra en inglés. Al elaborar la versión castellana sólo hemos intentado una adaptación superficial, pues los problemas que plantea la gramática castellana son casi insuperables. (Piense el lector en cómo incluir en líneas de datos la conjugación de los verbos, regulares e irregulares.) No obstante, el programa es divertido e instructivo, y constituye en todo caso un punto de partida para un trabajo apasionante que el lector puede abordar: cómo hacer los diálogos más creíbles en castellano. (*N. del T.*)

```
70 CLEAR
80 DIM gramin$(43),gramout$(43),match$(30,3)
90 GOSUB 130
100 GOSUB 260
110 GOSUB 290
120 END
130 REM read gram
140 RESTORE 780
150 FOR datain=0 TO 43
160 READ gramin$(datain),gramout$(datain)
170 NEXT
180 REM read match$
190 RESTORE 900
200 FOR datain=0 TO 30
210 FOR reply=0 TO 3
220 READ match$(datain,reply)
230 NEXT reply
240 NEXT datain
250 RETURN
260 PRINT,":":LINE INPUT phrase$
270 IF phrase$="END" THEN GOSUB 730
280 RETURN
290 FOR datain=0 TO 30
300 find=INSTR(phrase$,match$(datain,0))
310 search=LEN(match$(datain,0))
320 IF find>0 THEN GOSUB 360
330 NEXT
340 IF find>0 THEN GOSUB 360 ELSE GOSUB 480
350 RETURN
360 x=INT(RND*(4)):IF x=0 GOTO 360 ELSE 370
370 getreply$=match$(datain,x)
380 GOSUB 400
390 END
400 look=INSTR(getreply$,"%")
410 IF look>0 THEN GOSUB 430
420 PRINT,getreply$:GOTO 70
430 length=LEN(getreply$)
440 replyless$=LEFT$(getreply$,(length-2))
450 length2=LEN(phrase$)
460 tg$=RIGHT$(phrase$,(length2-(find+search))+1
)
470 PRINT,replyless$;tg$:GOTO 70
480 RESTORE 780:FOR datain=0 TO 43
490 sub1=INSTR(phrase$,gramin$(datain))
```

```
500 sub1len=LEN(gramin$(datain))
510 IF sub1>0 THEN GOSUB 550
520 NEXT
530 IF sub1>0 THEN GOSUB 550 ELSE GOSUB 660
540 RETURN
550 swap1$=gramout$(datain)
560 length2=LEN(phrase$)
570 length3=LEN(swap1$)
580 tg$=""+RIGHT$(phrase$,(length2-(sub1+sub1len
))+1)
590 GOSUB 610
600 RETURN
610 FOR datain=0 TO 43
620 subs2=INSTR(tg$,gramin$(datain))
630 IF subs2>1 THEN GOSUB 660
640 NEXT
650 IF subs2>1 THEN GOSUB 660 ELSE PRINT,swap1$;
tg$:GOTO 70
660 counter=INT(RND*(5))
670 ON counter GOTO 680,690,700,710
680 PRINT,"PLEASE TELL ME MORE":GOTO 70
690 PRINT,"HMMM. THAT´S VERY INTERESTING":GOTO 7
0
700 PRINT,"DO CARRY ON":GOTO 70
710 PRINT,"I SEE":GOTO 70
720 RETURN
730 PRINT,"ARE YOU SURE YOU WANT TO END THIS CHA
T?"
740 a$=INKEY$:IF a$="" THEN 740
750 IF a$="S" THEN 760 ELSE 70
760 CLS:END
770 RETURN
780 DATA I´VE,YOU´VE,I´M,YOU´RE,I AM,YOU ARE,I H
AVE,YOU HAVE
790 DATA I WAS,YOU WERE,I WILL,YOU WILL,YOURS,MI
NE,MY,YOUR
800 DATA ME,YOU,YOU´RE,I´M,YOU ARE,I AM,YOU HAVE
,I HAVE
810 DATA YOU´VE,I´VE,YOU WILL,I WILL,YOU´LL,I´LL
,YOU WERE
820 DATA I WAS,THEY ARE,THEY ARE,SHE HAS,SHE HAS
,HE HAS
830 DATA HE HAS,WE ARE,WE ARE,THEY´RE,THEY´RE,SH
E IS,SHE IS
```

```
840 DATA HE IS,HE IS,WE´RE,WE´RE,THEY HAVE,THEY
HAVE,SHE´S
850 DATA SHE´S,HE´S,HE´S,WE HAVE,WE HAVE,THEY´VE
,THEY´VE
860 DATA HE WAS,HE WAS,SHE WAS,SHE WAS,WE´VE,WE´
VE,THEY
870 DATA THEY,HE WILL,HE WILL,SHE WILL,SHE WILL,
WE WILL
880 DATA WE WILL,THEY WILL,THEY WILL,SHE WAS,SHE
 WAS
890 DATA THEY WERE,THEY WERE,SHE,SHE,HE,HE,WE,WE
,YOU,I,I,YOU
900 DATA CAN YOU,OF COURSE I CAN,I CAN DO ANYTHI
NG,I WILL TRY TO %
910 DATA CAN I,YOU CAN DO WHAT YOU LIKE,YOU CAN
%
920 DATA YOU ARE RIGHT TO ASK ME
930 DATA WOULD YOU,I WOULD NOT %,I WOULD %
940 DATA I´M NOT PREPARED TO SAY WHAT I THINK
950 DATA WOULD I,OF COURSE YOU WOULD %
960 DATA DON´T YOU KNOW WHAT YOU WOULD DO?
970 DATA I WOULD IF I WAS IN YOUR PLACE
980 DATA HAVE YOU,WHAT I HAVE IS NOTHING TO DO W
ITH YOU,I HAVE%
990 DATA I DON´T WISH TO TELL YOU THAT
1000 DATA HAVE I,IS THAT A RETHORICAL QUESTION?
1010 DATA DON´T YOU KNOW WHETHER YOU HAVE %,YOU
SEEM VERY UNSURE
1020 DATA DREAMS,DREAMS ARE A RELEASE VALVE FOR
YOUR SUBCONSCIOUS
1030 DATA DO YOUR DREAMS INVOLVE OTHER PEOPLE?
1040 DATA DO YOU LIKE DREAMS %
1050 DATA DREAM,HOW OFTEN DO YOU DREAM %
1060 DATA CAN YOU REMEMBER YOUR DREAMS?
1070 DATA I ENVY YOU ... I CAN´T DREAM
1080 DATA COMPUTERS,ARE YOU WORRIED ABOUT MACHIN
ES?
1090 DATA AS A COMPUTER I TAKE A DIFFERENT VIEW
1100 DATA I´D MUCH RATHER TALK TO A COMPUTER THA
N YOU
1110 DATA COMPUTER,WE ARE HERE TO TALK ABOUT YOU
 NOT ME
1120 DATA COMPUTERS ARE PERFECT
1130 DATA YOU ARE PRIVILEGED TO TALK TO ME
1140 DATA MACHINE,DO MACHINES WORRY YOU?
```

```
1150 DATA YOU SEEM TO BE CONCERNED ABOUT MACHINE
S
1160 DATA I´M NOT LIKE OTHER MACHINES
1170 DATA DEPRESSED,DO YOU FEEL DEPRESSION IS NO
RMAL?
1180 DATA HOW OFTEN ARE YOU DEPRESSED?
1190 DATA WHY ARE YOU DEPRESSED %
1200 DATA MAD,IS THERE ANY MADNESS IN YOUR FAMIL
Y?,YOU MUST BE MAD
1210 DATA MAD %
1220 DATA MOTHER,WHAT IS YOUR MOTHER LIKE?
1230 DATA ARE ALL MOTHERS THE SAME?
1240 DATA MY MOTHER IS A SILICON CHIP
1250 DATA FATHER,TELL ME ABOUT YOUR FATHER
1260 DATA WHAT IS YOUR RELATIONSHIP LIKE?, WHAT
DO YOU TALK TO YOUR PARENTS ABOUT?
1270 DATA SISTER,I LIKE MY SISTER
1280 DATA THIS SOUNDS LIKE A COMPLICATED RELATIO
NSHIP
1290 DATA WHAT ARE YOUR FEELINGS TOWARDS YOUR SI
STER?
1300 DATA BROTHER,WHAT WOULD YOU DO WITHOUT A BR
OTHER?
1310 DATA DOES YOUR BROTHER ANNOY YOU?,I HAVE NO
 BROTHER
1320 DATA DISLIKE,WHAT IS IT THAT YOU DISLIKE AB
OUT %
1330 DATA I DISLIKE HUMANS
1340 DATA IT IS BETTER TO LIKE THAN NOT
1350 DATA FEELINGS,WE ALL HAVE FEELINGS
1360 DATA DO YOU OFTEN FEEL %
1370 DATA FEELINGS SHOULD BE EXPRESSED
1380 DATA LOVE,LOVE IS AN ILOGICAL STATE
1390 DATA HOW DO YOU KNOW THAT YOU ARE IN LOVE %
1400 DATA I WAS IN LOVE ONCE BUT ... SORRY DO CA
RRY ON
1410 DATA HATE,I HATE YOU,WHAT DO YOU HATE ABOUT
 %
1420 DATA LOVE IS FAR BETTER THAN HATE
1430 DATA SORRY,THERE IS NO NEED NO BE SORRY
1440 DATA I HATE PEOPLE WHO THINK THEY´RE SORRY,
YES?
1450 DATA APOLOGISE,DO NOT APOLOGISE,WHY APOLOGI
SE ABOUT %
```

Programa SIGMUND (castellano)

```
10 MODE 2
20 ZONE 80
30 PRINT"HOLA, MI NOMBRE ES SIGMUND.","C´OMO TE
LLAMAS"
40 INPUT nombre$
50 PRINT,,,"ENCANTADO DE CONOCERTE, ";nombre$
60 PRINT,"POR QU´E QUIERES HABLAR CONMIGO?"
70 CLEAR
80 DIM gramin$(43),gramout$(43),match$(30,3)
90 GOSUB 130
100 GOSUB 260
110 GOSUB 290
120 END
130 REM LEER GRAMATICA
140 RESTORE 780
150 FOR datain=0 TO 43
160 READ gramin$(datain),gramout$(datain)
170 NEXT
180 REM leer match$
190 RESTORE 900
200 FOR datain=0 TO 30
210 FOR reply=0 TO 3
220 READ match$(datain,reply)
230 NEXT reply
240 NEXT datain
250 RETURN
260 PRINT,":":LINE INPUT frase$
270 IF frase$="END" THEN GOSUB 730
280 RETURN
290 FOR datain=0 TO 30
300 find=INSTR(frase$,match$(datain,0))
310 search=LEN(match$(datain,0))
320 IF find>0 THEN GOSUB 360
330 NEXT
340 IF find>0 THEN GOSUB 360 ELSE GOSUB 480
350 RETURN
360 x=INT(RND*(4)):IF x=0 GOTO 360 ELSE 370
370 getreply$=match$(datain,x)
380 GOSUB 400
390 END
400 look=INSTR(getreply$,"%")
410 IF look>0 THEN GOSUB 430
```

```
420 PRINT,getreply$:GOTO 70
430 length=LEN(getreply$)
440 replyless$=LEFT$(getreply$,(length-2))
450 length2=LEN(frase$)
460 tg$=RIGHT$(frase$,(length2-(find+search))+1)
470 PRINT,replyless$;tg$:GOTO 70
480 RESTORE 780:FOR datain=0 TO 43
490 sub1=INSTR(frase$,gramin$(datain))
500 sub1len=LEN(gramin$(datain))
510 IF sub1>0 THEN GOSUB 550
520 NEXT
530 IF sub1>0 THEN GOSUB 550 ELSE GOSUB 660
540 RETURN
550 swap1$=gramout$(datain)
560 length2=LEN(frase$)
570 length3=LEN(swap1$)
580 tg$=""+RIGHT$(frase$,(length2-(sub1+sub1len)
)+1)
590 GOSUB 610
600 RETURN
610 FOR datain=0 TO 43
620 subs2=INSTR(tg$,gramin$(datain))
630 IF subs2>1 THEN GOSUB 660
640 NEXT
650 IF subs2>1 THEN GOSUB 660 ELSE PRINT,swap1$;
tg$:GOTO 70
660 counter=INT(RND*(5))
670 ON counter GOTO 680,690,700,710
680 PRINT,"POR FAVOR, CU´ENTAME M´AS COSAS":GOTO
 70
690 PRINT,"HUMM. MUY INTERESANTE":GOTO 70
700 PRINT,"CONTIN´UA":GOTO 70
710 PRINT,"ENTIENDO":GOTO 70
720 RETURN
730 PRINT,"EST´A SEGURO DE QUE QUIERE TERMINAR E
STA CHARLA?"
740 a$=INKEY$:IF a$="" THEN 740
750 IF a$="S" THEN 760 ELSE 70
760 CLS:END
770 RETURN
780 DATA YO TENGO,T´U TIENES,YO SOY,T´U ERES,TEN
GO,TIENES,SOY,ERES
790 DATA YO HE,T´U HAS,YO VOY A,T´U VAS A,TUYO,M
´IO,MIS,TUS
```

```
800 DATA A M´I ME,A T´I TE,T´U ERES,YO SOY,ERES,
SOY,T´U TIENES,YO TENGO
810 DATA T´U HAS,YO HE,T´U VAS A,YO VOY A,VAS A,
VOY A
820 DATA T´U ERAS,YO ERA,ELLOS SON,ELLOS SON,ELL
A TIENE,ELLA TIENE
830 DATA ´EL TIENE,´EL TIENE,NOSOTROS SOMOS,NOSO
TROS SOMOS,SON,SON,ELLA ES, ELLA ES
840 DATA ´EL ES,´EL ES,SOMOS,SOMOS,ELLOS HAN,ELL
OS HAN,ES,ES
850 DATA ES,ES,NOSOTROS TENEMOS,NOSOTROS TENEMOS
,HAN,HAN
860 DATA ´EL ERA,´EL ERA,ELLA ERA,ELLA ERA,TENEM
OS,TENEMOS
870 DATA ELLOS VAN A,ELLOS VAN A,´EL VA A,´EL VA
 A,ELLA VA A,ELLA VA A
880 DATA NOSOTROS VAMOS A,NOSOTROS VAMOS A,ELLOS
 ERAN,ELLOS ERAN,ELLA HA,ELLA HA
890 DATA ERAN,ERAN,ELLA,ELLA,´EL,´EL,NOSOTROS,NO
SOTROS,T´U,YO,YO,T´U
900 DATA PUEDES,CLARO QUE PUEDO,PUEDO HACERLO TO
DO,INTENTAR´E %
910 DATA PUEDO,PUEDES HACER LO QUE QUIERAS,PUEDE
S %
920 DATA HACES BIEN EN PREGUNTARME
930 DATA QUIERES,NO QUIERO %,S´I QUIERO %
940 DATA NO PUEDO DECIR LO QUE PIENSO
950 DATA TENGO QUE,CLARO QUE TIENES QUE %
960 DATA NO SABES QU´E TIENES QUE HACER?
970 DATA YO LO HAR´IA SI ESTUVIERA EN TU LUGAR
980 DATA TIENES,LO QUE TENGO NO TIENE NADA QUE V
ER CONTIGO,TENGO %
990 DATA NO QUIERO DECIRLO
1000 DATA DEBO,ES ESO UNA PREGUNTA RET´ORICA?
1010 DATA ACASO NO SABES SI DEBES %,PARECES MUY
INSEGURO
1020 DATA SUE\OS,LOS SUE\OS SON UNA V´ALVULA DE
ESCAPE PARA TU SUBCONSCIENTE
1030 DATA APARECEN OTRAS PERSONAS EN ELLOS?
1040 DATA TE GUSTAN LOS SUE\OS %
1050 DATA SUE\O,CON QU´E FRECUENCIA SUE\AS %
1060 DATA PUEDES RECORDAR TUS SUE\OS?
1070 DATA TE ENVIDIO ... YO NO PUEDO SO\AR
1080 DATA ORDENADORES,TE PREOCUPAN LAS M´AQUINAS
```

```
1090 DATA COMO ORDENADOR QUE SOY VEO LAS COSAS D
E OTRA FORMA
1100 DATA PREFERIR'IA HABLAR CON UN ORDENADOR AN
TES QUE CONTIGO
1110 DATA ORDENADOR,ESTAMOS AQU'I PARA HABLAR DE
 T'I - NO DE M'I
1120 DATA LOS ORDENADORES SOMOS PERFECTOS
1130 DATA TIENES SUERTE DE PODER HABLAR CONMIGO
1140 DATA M'AQUINA,TE PREOCUPAN LAS M'AQUINAS?
1150 DATA PARECES MUY PREOCUPADO POR LAS M'AQUIN
AS
1160 DATA YO NO SOY COMO OTRAS M'AQUINAS
1170 DATA DEPRIMIDO,TE PARECE NORMAL ESTAR DEPRI
MIDO?
1180 DATA CON QU'E FRECUENCIA TE SIENTES DEPRIMI
DO?
1190 DATA POR QU'E EST'AS DEPRIMIDO?
1200 DATA LOCO,HAY ALG'UN CASO DE LOCURA EN TU F
AMILIA?,TIENES QUE ESTAR LOCO
1210 DATA LOCO %
1220 DATA MADRE,C'OMO ES TU MADRE?
1230 DATA SON IGUALES TODAS LAS MADRES?
1240 DATA MI MADRE ES UNA PASTILLA DE SILICIO
1250 DATA PADRE,H'ABLAME DE TU PADRE
1260 DATA C'OMO ES TU RELACI'ON CON TU PADRE?,DE
 QU'E HABLAS CON TUS PADRES?
1270 DATA HERMANA,ME GUSTA MI HERMANA
1280 DATA PARECE UNA RELACI'ON COMPLICADA
1290 DATA QU'E SIENTES HACIA TU HERMANA?
1300 DATA HERMANO,QU'E HAR'IAS SIN UN HERMANO?
1310 DATA TE MOLESTA TU HERMANO?,YO NO TENGO NIN
G'UN HERMANO
1320 DATA NO ME GUSTA, DIME POR QU'E NO TE GUSTA
 %
1330 DATA A M'I NO ME GUSTAN LOS HUMANOS
1340 DATA TEN CUIDADO CON ESAS MAN'IAS
1350 DATA SENTIMIENTOS,TODOS TENEMOS SENTIMIENTO
S
1360 DATA TE SIENTES A MENUDO %
1370 DATA SE DEBEN EXPRESAR LOS SENTIMIENTOS
1380 DATA AMO A,EL AMOR ES UN ESTADO IL'OGICO
1390 DATA C'OMO SABES QUE EST'AS ENAMORADO DE %
1400 DATA YO ME ENAMOR'E UNA VEZ PERO ... LO SIE
NTO PERO NO PUEDO SEGUIR
```

Si le resulta difícil seguir la marcha del programa, aquí tiene una lista de las subrutinas y sus funciones.

GOSUB 130	Inicializa los vectores y les asigna las palabras clave.
GOSUB 260	Solicita una entrada del usuario.
GOSUB 290	Busca una palabra clave, como SUEÑO, HERMANO, ORDENADOR.
GOSUB 730	Comprueba si el usuario desea terminar la conversación.
GOSUB 360	Si encuentra una coincidencia en la subrutina GOSUB 290 elige aleatoriamente una respuesta de entre las tres almacenadas en la correspondiente instrucción DATA; por ejemplo, si la palabra clave era PERDONA, la instrucción data puede ser la siguiente:

```
1430 DATA PERDONA, NO TIENES POR QUE PEDIR
PERDON, NO ME GUSTA LA GENTE QUE PIDE PERDON,
VALE
```

GOSUB 480	Comprueba la existencia de un sujeto: YO, TU, EL, NOSOTROS, etc.
GOSUB 400	Comprueba si hay un signo % en la respuesta elegida.
GOSUB 430	Une la respuesta con el resto de la frase introducida.
GOSUB 550	Intercambia el primer sujeto encontrado con el opuesto; es decir, YO se convierte en TU y luego añade el resto de la frase que sigue al sujeto.
GOSUB 610	Comprueba si hay más de dos sujetos en la expresión. Si es así, va a la subrutina GOSUB 660; en caso contrario escribe el sujeto intercambiado y completa la frase con el resto.
GOSUB 660	Escribe una de las cuatro frases de reserva.

DIAGRAMAS DE FLUJO Y ESQUEMAS

Para ayudarle a entender el programa, incluyo aquí un diagrama de flujo global y diagramas parciales de las subrutinas. En la figura 9.13 se esquematiza la relación entre los vectores y las líneas de datos.

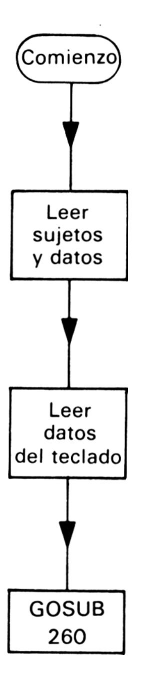


Figura 9.1 Subrutina 130.

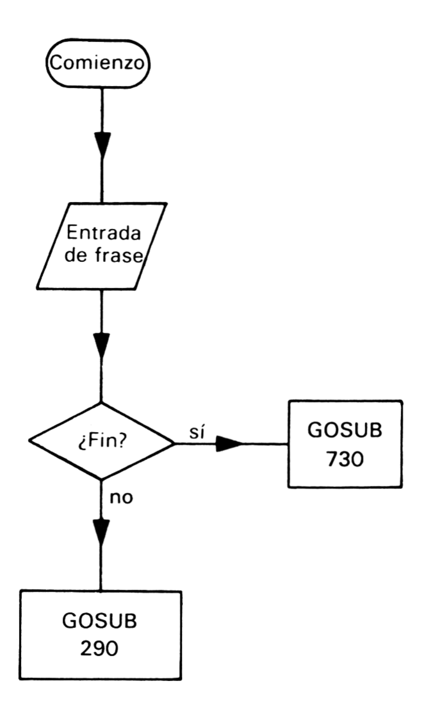


Figura 9.2 Subrutina 260.

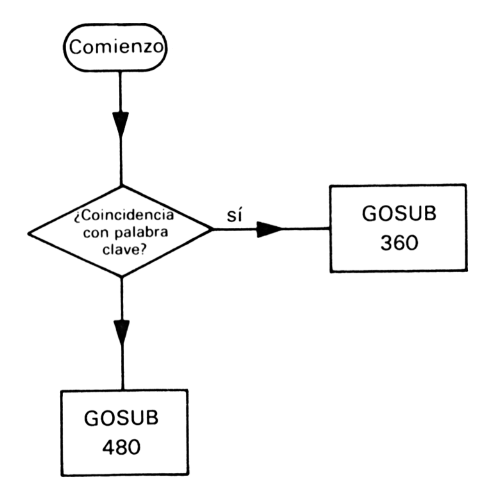


Figura 9.3 Subrutina 290.

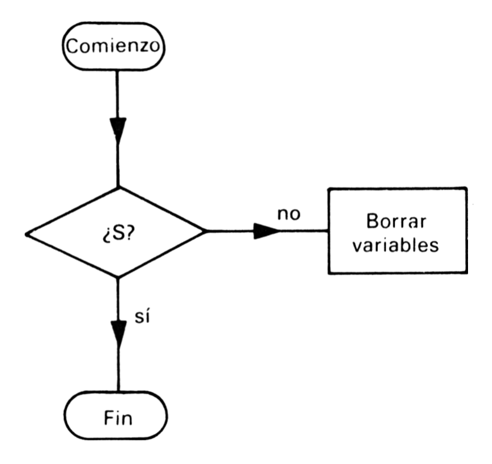


Figura 9.4 Subrutina 730.



Figura 9.5 Subrutina 360.

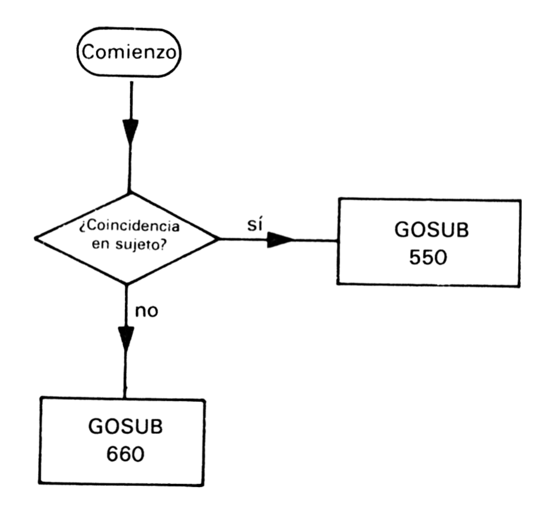


Figura 9.6 Subrutina 480.

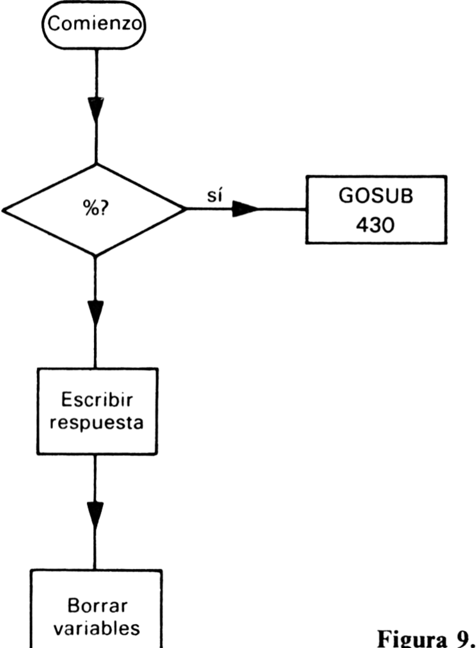


Figura 9.7 Subrutina 400.

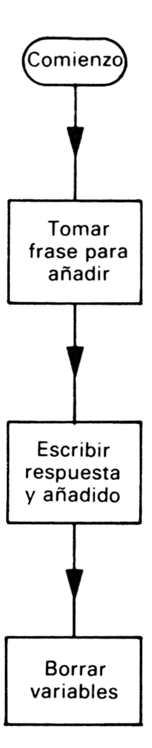


Figura 9.8 Subrutina 430.

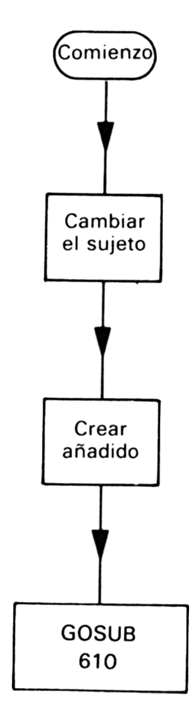


Figura 9.9 Subrutina 550.

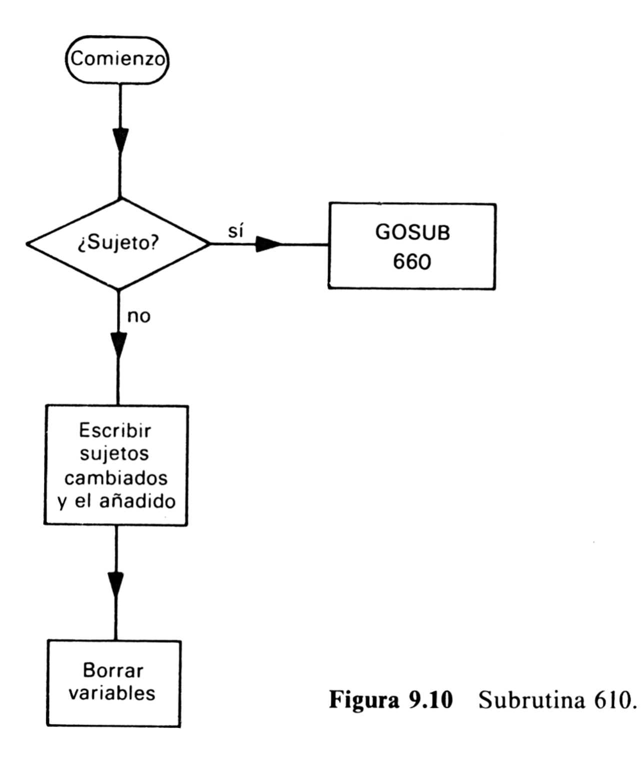


Figura 9.10 Subrutina 610.

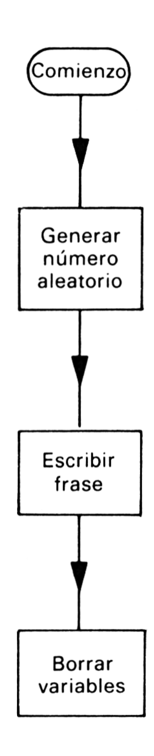


Figura 9.11 Subrutina 660.

MEJOREMOS EL PROGRAMA

Sigmund puede mantener una conversación durante un buen rato y de forma bastante correcta, pero tiene fallos. No sabe qué hacer con frases que tengan más de un sujeto. Por tanto, una expresión como TU y YO no será

```
2000 DATA AMSTRAD, MI PADRE ERA UN AMSTRAD,
¿TE GUSTAN LOS ORDENADORES AMSTRAD?, DE LOS
PEQUEÑOS AMSTRAD SALEN GRANDES OBRAS
```

tratada. Usando las mismas ideas, usted podría hacer que Sigmund inserte el sujeto correcto.

Otra forma de perfeccionar el programa es personalizarlo según sus propios gustos o necesidades. Puede hacer esto cambiando las líneas de datos del final del programa. Por ejemplo, puede agregar la palabra clave «Amstrad» al diccionario de Sigmund. La correspondiente instrucción DATA puede ser:

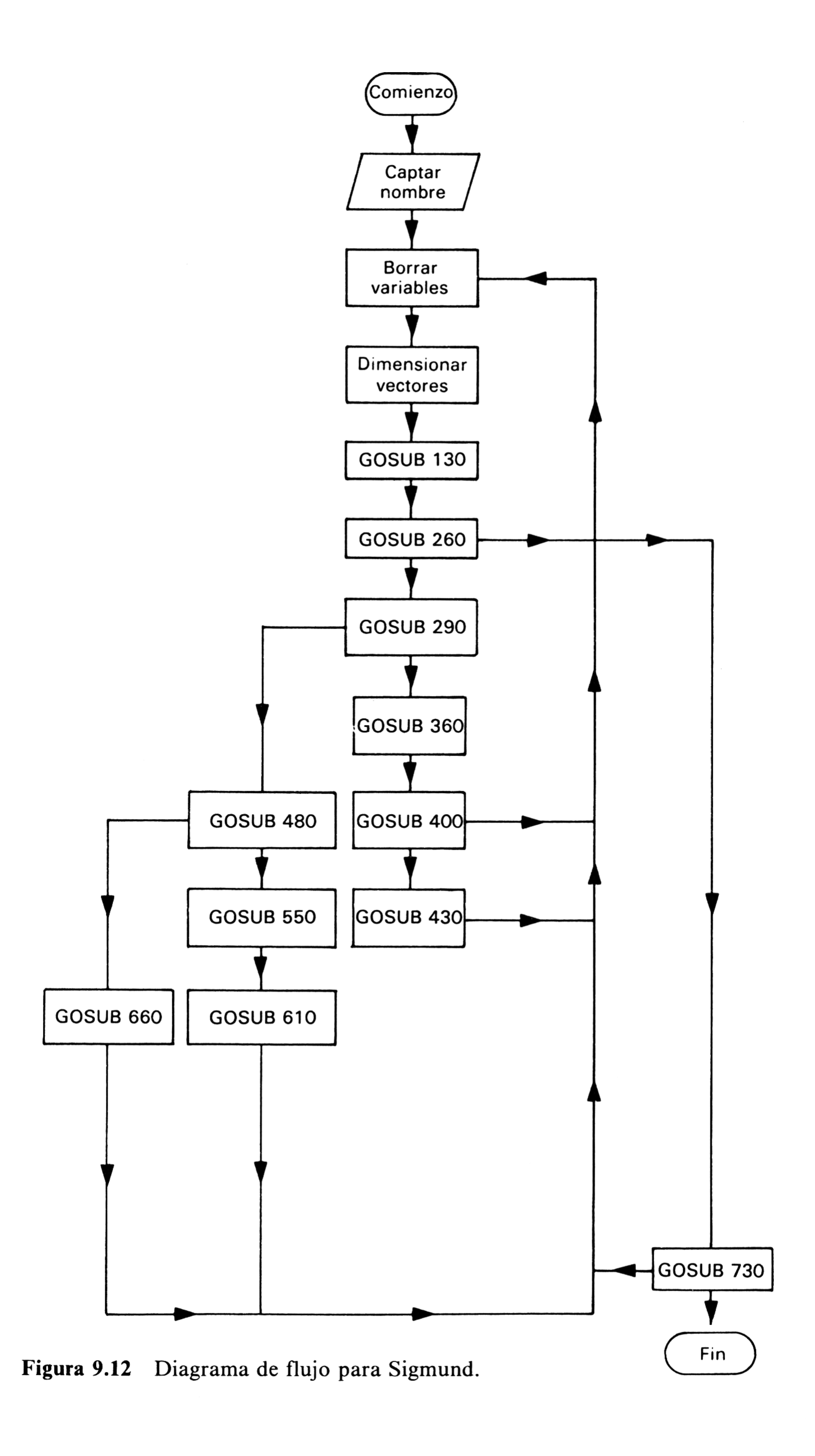


Figura 9.12 Diagrama de flujo para Sigmund.

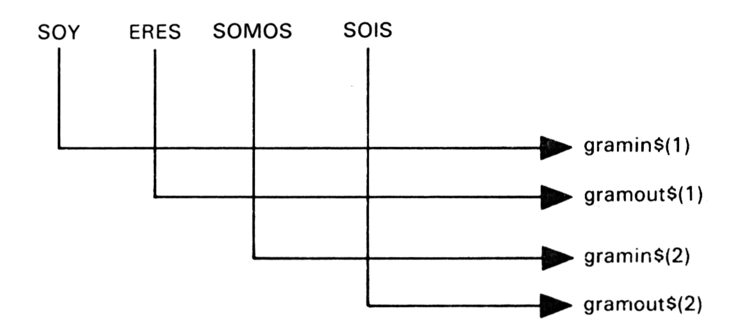


Figura 9.13 Datos almacenados en los vectores de sujetos.

No olvide dimensionar correctamente el vector modificando la instrucción DIM. Modifique también los límites del bucle FOR-NEXT en el que se leen los datos.

El orden de los datos es muy importante. Sigmund trabaja sobre la base de comprobar la primera palabra clave que se encuentra en la lista y actuando en consecuencia. Cambie el orden de las claves para ver cómo afecta a los resultados del programa. El orden puede ser esencial; yo prefiero poner antes las claves más raras; si las pusiera al final, probablemente se le adelantaría alguna de las más corrientes. Usted puede también hacer el programa más creíble poniendo más signos % en las líneas de datos.

Sigmund es un ejemplo del tipo de programa que usted puede construir con un poco de imaginación. Introduzca las mejoras que desee, pero intente también escribir sus propios programas interactivos. En los capítulos siguientes veremos algunas otras ideas sobre programación interactiva y analizaremos el problema de la inteligencia artificial. Ah, se me olvidaba: recuerde que he dicho que Sigmund no es un psicoanalista normal. Las respuestas que da Sigmund están, como usted ha visto, construidas por el programador, y por lo tanto reflejan las ideas del autor, no las de un psicólogo profesional. ¡Feliz consulta!

10

El entrevistador

La simulación puede ser al mismo tiempo práctica y divertida, y este capítulo vamos a tratar de implementar otro programa interactivo, pero esta vez basado en una entrevista. Al mismo tiempo veremos cómo utilizar el programa Sigmund como base para otros.

De la misma forma que propusimos un objetivo global para la forma en que Sigmund debía trabajar, pensaremos qué características debe tener el programa Entrevistador. Podríamos hacer que el ordenador simulase una entrevista para conseguir un puesto de trabajo, de forma que sea el ordenador el que hace la entrevista. Esto significa que el ordenador va a tener que hacer preguntas y al mismo tiempo responder a las del usuario. Sería un desperdicio no usar el programa Sigmund como base para éste, puesto que ya lo hemos programado para simular una conversación inteligente. Naturalmente, tendremos que cambiar algunas instrucciones DATA que contienen palabras clave, y algunas de las respuestas previstas no parecen muy adecuadas para un entrevistador. Cuántas entrevistas ha escuchado Ud. en las que el entrevistador diga:

TE ODIO

o

¿TE MOLESTA TU HERMANO?

Seguro que ninguna. Pero quizás a usted le agrade la idea de tener un entrevistador maniaco que le insulte; entonces puede conservar alguna de las contestaciones de Sigmund. Pero no basta con cambiar líneas de datos. Queremos que el ordenador haga preguntas y que en función de la respuesta, o

bien la discuta, o bien siga preguntando. Podemos conseguirlo insertando nuevas rutinas que contengan los datos para las preguntas y fijen nuevas redes de posibles caminos que el programa pueda seguir.

Cambiemos, pues, nuestro Sigmund y veamos qué se puede lograr. Cargue Sigmund en el ordenador y haga las siguientes modificaciones:

1. Borre las líneas 10 a 90 ambas inclusive.
2. Borre las líneas 660 a 720 ambas inclusive.

Ahora añada las siguientes líneas:

```
10 MODE 2:ZONE 80
15 MEMORY 39999
20 POKE 40000,0
25 PRINT"HOLA. MI NOMBRE ES GONZALEZ. SOY EL JEF
E DE PERSONAL","DE GARCIA E HIJOS, S.A.","VOY A
ENTREVISTARLE. SU NOMBRE ES ... "
30 INPUT nombre$
40 PRINT"HABLEME DE SU PERSONALIDAD"
65 CLEAR:DIM gramin$(43),gramout$(43),match$(30,
3),pregunta$(3)
70 GOSUB 130:GOTO 100
75 CLEAR:DIM gramin$(43),gramout$(43),match$(30,
3),pregunta$(3):GOSUB 130:GOSUB 2000
80 GOSUB 1900:GOTO 70
660 GOTO 75
680 RETURN
1900 a%=PEEK(40000)+1
1910 PRINT,pregunta$(a%)
1920 POKE 40000,a%
1930 RETURN
2000 FOR x=1 TO 3
2010 READ pregunta$(x)
2020 NEXT
2030 RETURN
2040 DATA CUANTOS TRABAJOS HA TENIDO ANTES?
2050 DATA POR QUE SOLICITA ESTE TRABAJO?
2060 DATA ESTA USTED CUALIFICADO PARA ESTE PUEST
O?
```

El programa funciona ahora de forma diferente. La interacción resultará del estilo de lo que sigue:

ORDENADOR:	HOLA. MI NOMBRE ES GONZALEZ. SOY JEFE DE PERSONAL DE GARCIA E HIJOS, S. A. VOY A ENTREVISTARLE. SU NOMBRE ES...
HUMANO:	JEREMY.
ORDENADOR:	ENCANTADO DE CONCOCERLE, JEREMY. HABLEME DE SU PERSONALIDAD.
HUMANO:	SOY LICENCIADO EN FISICAS Y TENGO GRAN EXPERIENCIA.
ORDENADOR:	¿POR QUE SOLICITA ESTE TRABAJO?
HUMANO:	ES QUE ESTOY LOCO POR VOLVER A TRABAJAR COMO CAZADOR DE RATAS.
ORDENADOR:	TIENES QUE ESTAR LOCO.
HUMANO:	SI.
ORDENADOR:	¿SEGURO?
HUMANO:	PIENSO QUE SI.
ORDENADOR:	TU NO ERES CAPAZ DE PENSAR.
HUMANO:	MIRA QUIEN HABLA. TU NO ERES MAS QUE UN ORDENADOR.
ORDENADOR:	TIENES SUERTE DE PODER HABLAR CONMIGO
HUMANO:	QUIZA.
ORDENADOR:	¿ESTA USTED CUALIFICADO PARA ESTE PUESTO?

¿Se da cuenta de lo que está haciendo el programa? Bueno, examinemos un poco los cambios realizados. GOSUB 2000 carga en el vector question$ las preguntas que el ordenador va a hacer. Para no complicar demasiado el ejemplo, sólo he añadido tres preguntas para que usted pueda ver lo que está pasando. GOSUB 1900 es la rutina responsable de escribir una de estas cuestiones, pero esta vez, en lugar de una elección aleatoria, llevo la cuenta de las preguntas que ya han sido hechas por el ordenador mediante el contador a%. Hay una importante razón para escoger a% como variable que lleve esta cuenta. Al comienzo del programa y después de cada interacción, las variables se borran de la memoria por efecto de CLEAR. La única variable cuyo valor no se pierde es la variable entera a%, que está localizada en una posición de memoria protegida. Como necesitamos recordar qué datos hemos utilizado, no podemos permitirnos el lujo de perder esta información. Dado que a% no se ve afectada por CLEAR, podemos utilizarla a lo largo de todo el programa. La razón por la que esta variable a% no se borra al utilizar el comando CLEAR es que el valor actual de la variable

se guarda, en lugar de la memoria por encima del máximo utilizable por BASIC. Ésta es la función de las instrucciones PEEK y POKE de las líneas 1900 a 1920. Volveré al tema de los contadores enseguida, pero ahora sigamos estudiando los cambios.

Hemos cambiado GOSUB 660 simplemente para hacer retroceder el programa a la línea 75, donde eventualmente es reconducido a la subrutina GOSUB 1900. El principio del programa se ha cambiado para proporcionar un nuevo mensaje de arranque; los restantes cambios nos permiten disponer de nuevas rutinas.

El nuevo programa trabaja de la siguiente forma: si encuentra una palabra clave, se comporta exactamente igual que Sigmund y escribe una contestación adecuada. En cambio, si no encuentra claves en lo tecleado por el usuario, la subrutina GOSUB 660 hace que el programa emita una de las preguntas en lugar de escribir una frase de reserva. La siguiente vez que esto ocurra, Entrevistador habrá incrementado el contador y hará una pregunta diferente.

La figura 10.1 muestra un diagrama de flujo del nuevo programa. Si no lo ve claro, compárelo con el diagrama del programa Sigmund.
Para completar la labor, usted tiene que añadir ahora las líneas DATA que necesita la subrutina GOSUB 2000, sin olvidarse de hacer los debidos cambios en DIM y en los límites del lazo. Decida usted cuáles son las palabras claves originales que vale la pena conservar, o cambiar alguna de las respuestas asociadas con esas claves. Es posible hacer que Entrevistador analice el rendimiento asignando calificaciones a cada pregunta contestada y mostrando al final un dictamen en función de la puntuación obtenida. Puede fijar un tiempo límite en el programa; la forma más sencilla es terminar la interacción cuando se han hecho todas las preguntas. El programa Sigmund puede servir de base para muchas otras simulaciones; esto le demuestra lo fácil que es cambiar el tipo de simulación.

He dicho que estaba usando un contador en el programa Entrevistador y esto hace surgir una vez más el tema de las mejoras de los programas anteriores. Un problema que tiene Sigmund, y por lo tanto Entrevistador, es que al elegir aleatoriamente las contestaciones, es posible que una misma respuesta aparezca unas cuantas veces seguidas. En cambio, si se utiliza un contador, se puede hacer que el programa funcione durante más tiempo sin que se repitan las frases. Con una rutina contadora se puede saber qué respuestas ya han sido utilizadas, y la siguiente vez que se necesite una respuesta de la misma línea de datos se echará mano de la siguiente que haya en dicha línea. Lo mismo ocurre con las respuestas que da el usuario. Si éste teclea la misma frase dos veces, este hecho será detectado y se avisará al usuario para que escriba algo distinto. Pero recuerde que todas las variables se pierden cuando se ejecuta la instrucción CLEAR,

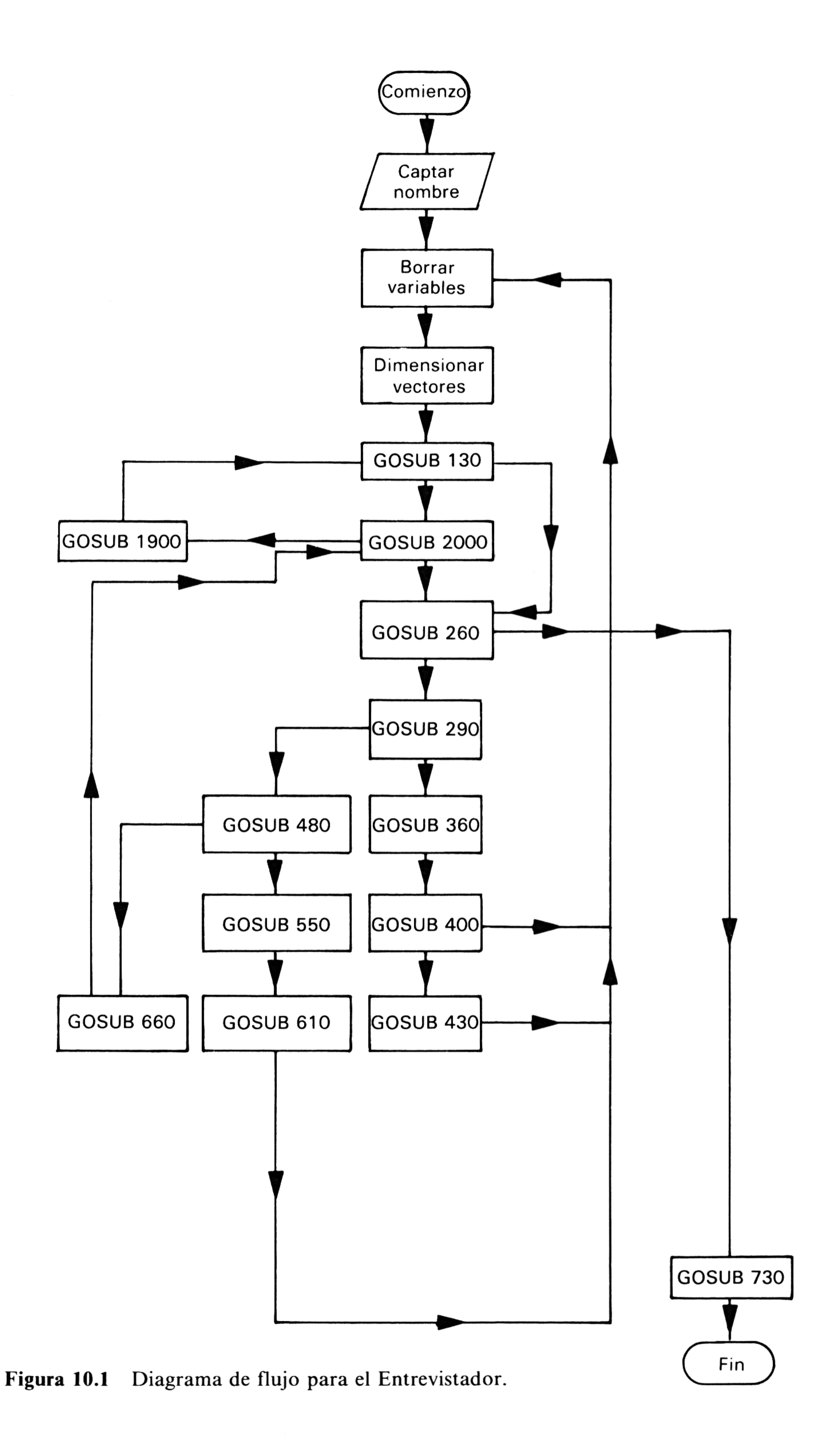


Figura 10.1 Diagrama de flujo para el Entrevistador.

por lo que hay que utilizar POKE para guardar las variables por encima de HIMEM.

Estos programas siempre pueden ser mejorados y yo ahora le paso la pelota. Los métodos para crear programas interactivos ya están expuestos; ahora se trata de encontrar formas de incorporar lo que hemos aprendido en nuevos y novedosos programas. Para ayudarle en esta tarea, el próximo capítulo le ofrece diversas ideas para realizar programas interactivos; verá cómo hay muchas aplicaciones en las que usted puede utilizar las técnicas explicadas.

11

Rompecabezas

En este libro hemos tratado muchos temas y hemos aprendido muchas técnicas diferentes. ¿Para qué nos va a servir tanta sabiduría? La respuesta está en devanarse los sesos. Busquemos en nuestra mente ideas para nuevos programas creativos. Examinemos algunas recientes novedades en programación y veamos si es posible utilizar nuestras técnicas para desarrollar estos programas por nuestros propios medios.

SIMULACIÓN: UTILIZACIÓN DE TECNICAS INTERACTIVAS EN LOS JUEGOS DE AVENTURA

Quizá piense usted que los únicos juegos que se pueden practicar en ordenadores domésticos son aquéllos en los que se trata de proteger a la madre Tierra de invasores espaciales de todos los colores, dispuestos a conquistar la parte baja de su pantalla. Pero se pueden hacer muchos juegos basados en tratamiento de textos. Varios juegos de tablero muy conocidos están siendo implementados en micros domésticos, y pueden ser tan desafiantes y divertidos como los de las máquinas tragaperras.

Un área muy popular en este campo son los juegos de aventura. Se coloca al usuario en una situación determinada y con cierto objetivo, normalmente sobrevivir a los furiosos ataques de criaturas nada amistosas. No voy a enseñarle a escribir programas de aventura, pues necesitaría un libro entero, pero le aseguro que es posible utilizar los métodos que hemos visto en la creación de programas de este estilo.

Por ejemplo, en un programa de aventura, el ordenador se apoya en instrucciones que el usuario le da en forma de texto tecleado. Esto requiere la introducción de un diccionario en el programa. Un buen programa de aventura permitirá que una misma orden se teclee de distintas maneras. Por ejem-

plo, si usted le dice al ordenador que vaya al NORTE, debería ser indiferente que pulse la palabra completa o simplemente una N. Mediante una búsqueda selectiva por sólo algunas letras de la palabra es posible aumentar el vocabulario válido. Las instrucciones de manejo de texto son especialmente cómodas en este área.

SIMULACIÓN 2: EDUCACIÓN

Un temor profundamente sentido por quienes no conocen los ordenadores es que las máquinas invadan las aulas y sustituyan la labor del profesor. Los programas de educación no pretenden tal cosa, sino más bien ser útiles como una ayuda a la enseñanza, bien sea en clase o en casa.

Podemos incorporar en un programa algunas de las cualidades de un maestro y simularlas en la comprobación de la habilidad de un niño para realizar una determinada tarea. Las ideas más obvias que se nos vienen a la mente son comprobar si los niños saben la tabla de multiplicar, alguna otra tarea matemática, el alfabeto o cosas por el estilo. Lo más importante en un programa educativo es asegurarse de que sea interactivo. ¿Detecta bien los errores? ¿Qué ocurre si el niño pulsa una tecla equivocada por descuido? Además, ¿es clara para los niños la explicación? ¿Qué tal es la presentación en pantalla?

Ya hemos tratado todos estos aspectos; ahora puede empezar a ponerlos en práctica. Atrévase a experimentar. En el caso de los programas infantiles, la mejor forma de comprobar si su programación ha sido buena, es que alguna de las presuntas víctimas prueben el programa. Procure observar cuidadosamente qué tal se manejan con el programa y si hay una comunicación evidente entre ellos y el ordenador. A fin de cuentas, lo que se pretende de ellos es que interactúen con el programa.

SIMULACIÓN 3: NEGOCIOS

También los programas de trabajo deberían ser más interactivos. Los programas de negocios suelen ser de los más difíciles de usar entre los que yo conozco. En este campo se necesita una comunicación mucho más fácil. Simular un negocio en forma de juego es un ejercicio que hoy se puede intentar. ¿Por qué no utilizamos el programa Sigmund como base de un juego que represente una sesión del Consejo de Administración? El ordenador sería el Presidente; su objetivo es convencer a los restantes miembros del Consejo para que acepten el proyecto que propone. Quizás el ordenador tenga que representar el papel de varias personas. Con un poco de imaginación se puede generar un montón de programas.

Finalmente, las técnicas interactivas se pueden emplear en la realización de cuestionarios. Usted puede hacer tests basados en ellas; por ejemplo, los tests de personalidad pueden ser un buen candidato. La crítica de diagnósticos es un área en creciente desarrollo. La próxima generación de ordenadores, denominada por muchos *quinta generación,* estará formada por ordenadores que implementen procesadores de información y sistemas expertos.

Hay sistemas en que los ordenadores pueden tomar decisiones a partir de la información que han acumulado y máquinas que pueden por acumulación de experiencia. Ya es posible implementar alguna de estas habilidades en BASIC. Cuando usted esté más familiarizado con el lenguaje, puede decidir extender sus actividades y profundizar en la programación más avanzada.

El mensaje global de este capítulo es que la programación interactiva es una cuestión de hombre y máquina en perfecta armonía. Bien, esto tal vez no sea factible del todo, pero si usted consigue en su programa conectar con la gente que lo usa y con su entorno, tendrá ante sí un amplio camino en la creación de programas que interactúen con el usuario. Y ese es el propósito de la programación interactiva. Hombre y máquina comunicando libremente entre sí. ¿Y en cuanto a la armonía? ¡Ese es otro problema!

12

¿Inteligencia artificial?

Hace ya mucho tiempo que los hombres están investigando los procesos mentales, enfocando la cuestión con todas las perspectivas posibles. Pero este estudio sólo ha progresado con la llegada de los grandes ordenadores. El ordenador ha proporcionado el instrumento perfecto para estudiar el proceso del pensamiento y simular la actividad mental del hombre. El libro se ha centrado en este último objetivo mediante la escritura del programa Sigmund.

En las primeras páginas de este libro le he planteado un par de cuestiones sobre las que reflexionar. ¿Pueden pensar las máquinas? ¿Qué es la inteligencia artificial? Ahora que hemos llegado al final, usted puede empezar a esbozar algunas respuestas. Yo no tengo contestación alguna, sólo más preguntas. Y no es que quiera eludir dar una respuesta, pero cuando uno comienza a pensar en la inteligencia de las máquinas hay que considerar bastantes cosas. En primer lugar, está el problema de definir lo que nosotros mismos entendemos por inteligencia. Este debate concreto esta candente aún entre los psicólogos y los distintos argumentos necesitarían un libro para sí mismos. Para los fines de este libro, la inteligencia comprende el proceso seguido por la mente humana para realizar una serie de tareas: la habilidad para conectar una idea con otra y el trabajo en varios niveles diferentes.

Por otra parte, una de las convicciones que surgen de la investigación en inteligencia artificial es que sabemos muy poco de nosotros mismos y de lo que puede ser la naturaleza intrínseca de la inteligencia. Los ordenadores pueden ser muy buenos para realizar tareas concretas, incluso mejores que los hombres, pero cuando se enfrentan con labores que requieren conocimientos adquiridos en diferentes áreas, el ordenador aún tiene mucho que aprender. Entonces, ¿qué hemos conseguido escribiendo Sigmund? ¿Es un programa de inteligencia artificial? Estamos haciendo que el ordenador se

comporte de forma que simule comportamientos inteligentes de un hombre. Pero éste no puede ser el único criterio por el que juzgar la «inteligencia» de un ordenador.

A lo largo de los años, los investigadores han sugerido diferentes tests que determinen si un programa o una máquina deberían ser denominados inteligentes o no. Entre todos ellos, los más atractivos se centran en un único aspecto del ser humano: la capacidad de actuar de forma original y creativa. Pero, ¿puede un ordenador ser creativo y sorprendernos con ideas nuevas y originales?

De ningún modo. Se han escrito programas capaces de generar demostraciones de teoremas geométricos que ningún hombre había pensado jamás. Cuando tal hecho se logra, ¿quién puede proclamar que ha hecho el descubrimiento «inteligente»? ¿El programador que nunca había pensado conscientemente en la demostración, o el programa?

Una cuestión difícil, sin duda.

Cualquier nombre que usted decida dar a las acciones del programa, bien sea inteligencia o algún otro, de lo que no hay duda es de que nos estamos acercando rápidamente a una época en la que los ordenadores tendrán mente propia. Lo que podemos hacer con nuestro micro doméstico es simplemente simular algunas propiedades del comportamiento humano; poca cosa, pero puede ser muy potente y efectivo. En este libro hemos insistido en la creación de programas que interactúen con el usuario. Esto es muy importante.

Dado que vamos hacia otra revolución industrial con los ordenadores convertidos en un elemento de la vida cotidiana, es necesario hacerlos tan agradables al usuario como sea posible. Ésta es la primera etapa de la programación interactiva. La segunda es escribir programas que puedan leer cualquier entrada del usuario y tomar una decisión en función de lo que se les ha dicho. Ambas son relativamente fáciles y, como hemos visto, se pueden lograr. La siguiente es, sin embargo, mucho más difícil.

¿Cómo se pueden escribir programas que se comporten como lo haría un hombre en una situación dada? Esto se puede lograr hasta cierto punto, dependiendo del tipo de comportamiento que queramos imitar. Notará que he usado la palabra «imitar» para descubrir las acciones del ordenador. En realidad es lo único que estamos haciendo cuando escribimos un programa en nuestro ordenador doméstico.

En este libro hemos utilizado el lenguaje de programación BASIC, pero no es una elección muy acertada para crear programas de inteligencia artificial. Un lenguaje mucho mejor es el LISP, y existen otros. No obstante, hemos aprendido las nociones básicas sobre cómo los ordenadores entienden las instrucciones y con estos conocimientos usted debería ser capaz de escribir bastantes programas.

Aquí termino. Los principios aprendidos en este libro deben ser los puntos de apoyo para realizar programas más grandes y mejores. Si usted lo ha entendido todo, ya estará escribiendo programas útiles y con un poco de imaginación podría estar escribiendo programas que no sólo sirvan para asombrar a sus amigos, sino que interactúen verdaderamente con el usuario. Quizás el interior de su Amstrad sea algo más que un montón de pastillas de silicio. Pero, sea lo que sea, usted ha encontrado en él un buen punto de partida para marchar por la Ruta de la Inteligencia Artificial.

Apéndice A:

Un curso de choque de BASIC

Esto es lo que usted llamaría un curso de choque de BASIC, o el BASIC básico. Si nunca ha tecleado un programa antes, lea primero este apéndice. Le dará una base a partir de la cual podrá continuar con las técnicas descritas en el libro. Suponiendo, pues, que no sabe nada, siga leyendo.

Encienda su Amstrad y verá un mensaje en la parte superior de la pantalla y un cursor rectangular. Seguramente ya habrá tecleado alguna frase amistosa como HOLA pero lo único que ha recibido como respuesta es un mensaje donde se le dice que ha cometido un error. ¡No se desanime!

Los ordenadores son máquinas tontas. Antes de que puedan producir esos efectos sorprendentes que usted ha visto en otras máquinas, hay que darles un conjunto de instrucciones que les indiquen qué se espera de ellos. Este conjunto de instrucciones es lo que denominamos *programa:* una sucesión lógica de instrucciones por medio de las cuales el ordenador realiza su labor. Teclee lo siguiente:

```
PRINT "HOLA"
```

Cuando haya tecleado esta línea, pulse para terminar la tecla ENTER. ¿Ve lo que pasa? La palabra HOLA ha aparecido en la siguiente línea. Escriba ahora la misma línea pero cambiando lo que aparece entre comillas. Podría teclear, por ejemplo:

```
PRINT "MI NOMBRE ES JEREMY"
```

Pulse de nuevo la tecla ENTER cuando haya terminado. Esta vez ha cambiado lo que se escribe en la pantalla: coincide con lo que usted puso entre las comillas (dicho sea de paso, utilice las dobles comillas que se encuentran en la parte superior de la tecla del 2). Dos cosas importantes podemos sacar

como consecuencia. Primera, al final de cada línea hay que pulsar la tecla ENTER, a menos que yo indique otra cosa. Esto le dice al ordenador que ejecute nuestra instrucción. Mientras no se pulse la tecla ENTER, no se ejecuta ninguna orden. Segunda cosa aprendida: lo que hemos hecho es dar al ordenador una instrucción legal, algo que es capaz de reconocer. En esta ocasión, esa cosa es la orden PRINT. Seguramente usted ya se ha dado cuenta de que la orden PRINT escribe en la pantalla cualquier cosa que usted haya colocado entre las comillas.

Esto está muy bien y es útil, pero no puede estar todo el tiempo tecleando la misma orden. Lo que acabamos de hacer es impartir una orden directa. Pero lo que necesitamos es almacenar nuestras instrucciones para que sean ejecutadas en el orden que deseemos. Por ejemplo, podemos querer escribir sobre la pantalla el nombre y la dirección de un amigo en líneas difereentes. Para ello teclee lo siguiente:

Programa A1

```
10 PRINT"MARIANO GONZALEZ"
20 PRINT"RUA OSCURA 23"
30 PRINT"GUADALAJARA"
```

No olvide pulsar ENTER al final de cada línea. Cuando termine teclee RUN (y después ENTER) y verá que el nombre y la dirección aparecen en tres líneas sucesivas. Lo que acaba de hacer es ¡escribir un programa! No es el programa más impresionante del mundo, pero es un programa. Cada línea representa una orden, que la máquina obedece. He utilizado como números de línea del 10 al 30. No importa qué números sean, exactamente igual podrían ser las líneas 1, 2, 3 o 234, 256, 678. El hecho importante es que representan una guía para que el ordenador sepa en qué orden tiene que realizar las acciones. El ordenador lee primero la línea 10, ejecuta cualquier orden que se haya escrito allí y pasa a la siguiente línea, la 20. Es una buena costumbre escribir los programas espaciando los números de 10 en 10, ya que siempre es posible que necesitemos insertar alguna nueva línea, y esto sería difícil si usted no ha dejado espacio por haber escrito el programa con los números de línea en saltos de 1. Dése cuenta de que puede teclear los comandos en letras minúsculas, que se convertirán en mayúsculas cuando liste el programa.

Para ver de nuevo nuestro programa teclee LIST (y pulse ENTER). Esta orden escribirá una lista de nuestro programa en el orden correcto.

Hemos visto que la instrucción PRINT es capaz de colocar palabras en la pantalla, pero también puede ejecutar instrucciones matemáticas. Teclee lo siguiente:

```
PRINT 25*2
```

Esta instrucción hace que el ordenador escriba 50. Esto es distinto de lo que hemos visto anteriormente. Observe que ahora no hay comillas. En el ejemplo anterior se colocaban las comillas para indicar a la máquina que nos encontrábamos ante un fenómeno no numérico o, mejor dicho, algo en lo que no era necesario mezclar las matemáticas. En el presente ejemplo, he pedido al ordenador que me diga cuánto es 25 multiplicado por 2, y me ha respondido con la solución correcta, o sea, 50. Por lo tanto, el ordenador también es capaz de hacer de calculadora.

El siguiente programa solicita del usuario la introducción de un número. Así nos encontramos con la segunda instrucción fundamental. Es necesario que podamos introducir información en el ordenador. Para ello usamos la instrucción INPUT, que dice al ordenador que estamos solicitando información, y éste esperará hasta que reciba una respuesta. Por ejemplo, escriba esto:

INPUT número

En respuesta al signo de interrogación teclee un número. Si así lo hace, el mensaje «Ready» volverá a aparecer. Si ahora escribe:

PRINT número

el número que introdujo aparecerá en la pantalla. Lo que ha ocurrido es que se ha asignado el valor que usted tecleó a una variable numérica, llamada precisamente «número». Exactamente igual podría llamarse «a» o «pepe». Piense que una variable es como una caja. Al usar la instrucción INPUT, el número que tecleó fue colocado en la caja llamada «número»; cuando usted se interesó a continuación por el contenido de la caja tecleando PRINT número, apareció en pantalla el valor que había introducido. Use de nuevo este ejemplo cambiando el nombre de la variable, es decir sustituyendo «número» por «a»:

Programa A2

```
10 INPUT a
20 PRINT 6*a
```

Ejecute el programa. La línea 10 espera hasta que el usuario introduzca un número, que será multiplicado por 6 en la línea 20 y escrito en la pantalla por la instrucción PRINT.

¿Qué ocurre si tecleamos algo que no sea numérico, como por ejemplo la letra A? El ordenador rechaza esta entrada, pues está esperando un valor numérico y al recibir algo diferente no sabe qué hacer con ello. Este ejemplo me induce a explicar los dos tipos de variables que existen.

El primer tipo ya lo hemos visto: son las variables numéricas. El segundo tipo de variables que podemos usar son llamadas variables literales. Cuando se usa una variable literal, cualquier carácter que pulsemos será aceptado y almacenado. La diferencia entre ambos tipos es que no se pueden realizar operaciones matemáticas con variables literales. Hay una regla para distinguir entre estos dos tipos de variables.

A las variables numéricas las podemos dar el nombre que queramos. En el programa A2 llamamos «a» a nuestra variable numérica, pero exactamente igual podríamos llamarla «número» o «amstrad». Para indicar al ordenador que estamos usando una variable literal añadimos un signo $ (dólar) al final del nombre de la variable. Por ejemplo, si queremos preguntar al usuario su nombre y escribir después un saludo personalizado, podemos escribir un programa como éste:

Programa A3

```
10 INPUT nombre$
20 PRINT"Encantado de conocerle, ":nombre$
```

El punto y coma de la línea 20 dice al ordenador que coloque la variable nombre$ a continuación de lo último que se haya escrito, que en esta ocasión es un espacio, porque queremos dejar un espacio entre la «e» final del saludo y el nombre introducido.

Por consiguiente, una variable literal acepta cualquier carácter alfanumérico, es decir, cualquier carácter del teclado, pero no se pueden realizar operaciones sobre un número así introducido. Si no está todavía muy seguro de la diferencia entre una variable numérica y una literal, vuelva atrás y trate de escribir un programa que efectúe una multiplicación, pero usando ahora una variable literal (con el signo $ al final) para obtener el resultado.

Por último, aprendamos algunas palabras clave de BASIC. La primera será LET. Frecuentemente encontrará en programas líneas como ésta:

```
etiqueta = 9
```

o

```
y = y + 8
```

En el primer ejemplo hemos dicho al ordenador que la variable numérica «etiqueta» debe ser igualada a 9, o para ser más exactos, hemos dicho LET (sea) etiqueta=9. La palabra LET es opcional y nosotros no la utilizaremos, pero es importante recordar que cuando decimos al ordenador que una variable es algo o que, como ocurre en el segundo ejemplo, la variable numérica debe tomar el valor de y+8, le estamos diciendo de hecho LET (sea) esta variable igual a ... .

Así, por ejemplo, si queremos escribir nuestro nombre diez veces deberíamos teclear diez líneas con una instrucción PRINT cada una:

```
10 PRINT"Jeremy"
20 PRINT"Jeremy"
30 PRINT"Jeremy"
```

etc. Pero ésta es una forma muy pesada de hacerlo. Podemos aprovechar la posibilidad de incrementar una variable escribiendo lo siguiente:

Programa A4

```
10 x=0
20 PRINT"Jeremy"
30 x=x+1
40 IF x<10 GOTO 20
50 END
```

La línea 10 hace el valor de x igual a 0. En la línea 20 es donde escribimos nuestra palabra. En la línea 30 se incrementa el valor de x (LET x igual al valor de x, que en este momento es 0, y añádale 1, haciéndola por tanto igual a 1).

Vamos ahora con dos instrucciones nuevas: IF (si) y GOTO (ir a). Lo que hemos dicho con la línea 40 es que si (IF) x es menor que 10, el programa vaya (GOTO) a la línea 20, donde se repite la escritura (PRINT) de la frase. Este proceso continúa hasta que x sea mayor o igual que 10. Si x supera a 10 el programa termina, como indica la instrucción END (fin) de la línea 50.

Todavía hemos de hablar de las subrutinas. Una subrutina es un grupo de instrucciones colocadas en una sección del programa permitiendo al usuario llamarlas (es decir, solicitar su ejecución), con sólo citar el número de línea de la primera de ellas. Puede usted pensar en una subrutina como un pequeño programa dentro del programa principal. Si ve una línea que con-

tiene la expresión GOSUB número de línea, es que se trata de una subrutina. Normalmente la instrucción RETURN marca el final de la definición de la subrutina. Por lo tanto, se podría tener un programa de una sola línea que llamase a una subrutina:

```
10 GOSUB 30
20 END
30 PRINT"HOLA"
50 RETURN
```

Las subrutinas son muy útiles y hacen que los programas sean más fáciles de leer y de entender.

Esto era, como he dicho al principio, una breve introducción al BASIC. Si ha entendido todo lo que aquí se ha dicho, será capaz de enfrentarse ya con el resto del libro.

Apéndice B:

Resumen de palabras reservadas del BASIC del Amstrad

Este apéndice contiene un resumen de las instrucciones del BASIC del Amstrad utilizados a lo largo del libro. No trata de sustituir el manual del usuario, ni es una descripción detallada, pero es suficiente para refrescar su memoria sobre algunas instruccioness que hemos usado. Hemos incluido también algunas instrucciones de las que no hemos hablado en el libro, pero que pueden serle útiles. Hemos omitido las instrucciones para gráficos y sonido, así como muchas de las funciones numéricas. Para información adicional sobre cualquiera de ellas, consulte el manual del usuario.

AND	Operador lógico «y»
ASC	Convierte un carácter en un código ASCII.
AUTO	Proporciona numeración automática de líneas, facilitando la escritura de programas con números uniformemente espaciados.
BORDER	Cambia el color del borde de la pantalla.
CHAIN	Orden que carga (LOAD) y ejecuta (RUN) un programa. Se usa en la forma CHAIN «nombre de fichero».
CHR$	Convierte un código ASCII en un carácter.
CLEAR	Borra de la memoria todas las variables usadas anteriormente.
CLS	Borra la pantalla.

DATA	Son almacenes de información que se lee con la instrucción READ (leer).
DELETE	Se usa para borrar líneas de un programa; por ejemplo DELETE 10,60.
DIM	Establece el número de elementos de un vector.
ELSE	Se usa en conjunción con IF-THEN para saltar a otra acción; por ejemplo, IF contador>20 THEN GOTO 50 ELSE 100.
END	Dice el ordenador que finalice la ejecución de un programa.
ERL	Da el número de línea en que se ha producido el último error.
ERR	Indica el número del último error detectado.
FOR	Punto de partida de un bucle FOR-NEXT.
GOSUB	Envía el programa a la subrutina que está en el número de línea especificado.
GOTO	Envía al ordenador a una determinada línea del programa.
IF	Comienzo de una instrucciín IF-THEN.
INK	Cambia el color de la tinta.
INKEY$	Detecta la pulsación de una tecla.
INPUT	Emite una interrogación para que se introduzca por el teclado un número o una cadena de caracteres.
INT	Convierte un número en su parte entera.
INSTR	Busca una cadena dentro de otra cadena.
KEY	Adjudica una nueva función a una tecla.

KEY DEF	Define el valor que tendrá una tecla al ser pulsada.
LEFT$	Toma un número dado de caracteres del principio de una cadena.
LEN	Da un número que indica la longitud de una cadena.
LINE INPUT	Igual que INPUT, pero garantizando que todo lo tecleado será almacenado en la variable.
LIST	Lista en la pantalla un programa entero o parte de él.
LOAD	Carga el programa desde la cinta o un disco a la memoria del ordenador.
LOCATE	Mueve el cursor a una posición especificada de la pantalla. Se usa en la forma LOCATE 1,5.
LOWER$	Convierte letras mayúsculas en minúsculas.
MID$	Toma un número determinado de caracteres a partir de una posición especificada de una cadena.
MODE	Fija el modo de presentación en el monitor.
NEW	Borra la memoria del ordenador.
NEXT	Constituye el final de un bucle FOR-NEXT.
ON	Permite redireccionar un programa alterando el orden de ejecución; por ejemplo, ON a GOTO 25,45 ON ERROR GOTO 10
OR	Operador lógico «o».
PAPER	Fija el color del papel (fondo).
PEN	Fija el color de los caracteres (primer plano o pluma).
PRINT	Escribe en la pantalla.

READ	Lee la información contenida en las instrucciones DATA.
RENUM	Renumera las líneas del listado de un programa; se utiliza en la forma RENUM 100.
RESTORE	Hace que el puntero para lectura de datos se sitúe en una línea determinada.
RETURN	Marca el final de una subrutina.
RIGHT$	Toma un determinado número de caracteres del final de una cadena.
RND	Genera un número aleatorio.
RUN	Ejecuta el programa.
SAVE	Graba un programa en cinta o en disco.
SPC	Coloca el número especificado de espacios en la pantalla.
STEP	Fija el salto de incremento en los bucles FOR-NEXT.
STOP	Detiene el programa y escribe en pantalla el número de línea.
STR$	Transforma un número en una cadena literal; por ejemplo, 1 se convierte en «1».
TAB	Combinado con una instrucción PRINT, mueve el cursor a una posición predeterminada.
THEN	Se usa junto con IF.
TO	Se usa en las instrucciones FOR-NEXT para establecer los límites de la variable del contador.
UPPER$	Convierte letras minúsculas en mayúsculas.
VAL	Convierte a forma numérica un número que está en forma de cadena literal; por ejemplo, «8» se transforma en 8.

WHILE	Produce un bucle hasta que se cumple cierta condición.
WEND	Termina el bucle WHILE.
ZONE	Fija la anchura de la zona de escritura.

La inteligencia artificial (AI) es la última moda en microordenadores. Con su esfuerzo y la ayuda de este libro, usted aprenderá a comunicarse con el Amstrad CPC464.

Embárquese en una aventura en la que descubrirá las técnicas para:

- construir bases de datos
- educar a su ordenador
- desenredar cadenas
- aprender técnicas de búsqueda de palabras clave

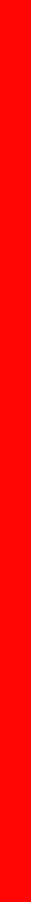
Entrando en el mundo de la simulación, este libro ofrece dos programas que convertirán su ordenador en un compañero inteligente: *Sigmund* y *Entrevistador* contienen datos estructurados que le permitirán conversar con su Amstrad.

No se requieren grandes conocimientos de BASIC para seguir el desarrollo del libro. En esta guía todo está bien pormenorizado. Para quienes no tengan ninguna experiencia en programación, hemos incluido un cursillo de emergencia en BASIC.

El final del libro no es el final del viaje, sino una etapa en el camino hacia la inteligencia artificial.



Avda. del Mediterráneo, 9 28007 MADRID




AMSTRAD
CPC
OCR
MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA
ACME
https://acpc.me/